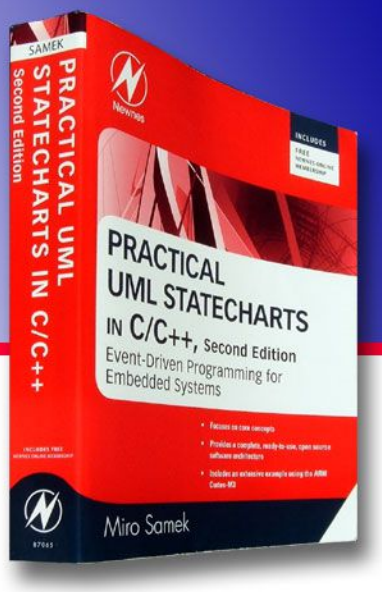
Quantum® LeaPs
innovating embedded systems



QDK™
Altera Nios II

Document Revision E
January 2013

Copyright © Quantum Leaps, LLC

www.quantum-leaps.com
www.state-machine.com

Table of Contents

1 Introduction 1
- 1.1 About QP™ 2
- 1.2 What's Included in the QDK-Nios-II? 2
- 1.3 Licensing QP™ 3

2 Installation 4

3 Creating Example Application and BSP 6
- 3.1 Launching the Nios II IDE 6
- 3.2 Configuring the FPGA 8
- 3.3 Debugging the Application 10
- 3.4 Creating the QP libraries (QEP, QF, QK, and QS) 13
- 3.5 Configuring the BSP 17
- 3.6 Configuring the Application 17
- 3.7 Using QS (Quantum Spy) Software Tracing 19
  - 3.7.1 Building the QS library 19
  - 3.7.2 Collecting the QS software trace 22

4 The Vanilla QP Port 24
- 4.1 The qf_port.h Header File 24
  - 4.1.1 The QF Critical Section 24
- 4.2 ISRs in the Non-preemptive "Vanilla" Configuration 25
  - 4.2.1 Registering Interrupt Handlers 25
  - 4.2.2 The Time-Tick Interrupt 25
  - 4.2.3 Optional Nesting of Interrupts 26
- 4.3 Idle Loop Customization in QF_onIdle() 26
- 4.4 Assertion Handling Policy in Q_onAssert() 27

5 The QK Port 28
- 5.1 The qk_port.h Header File 28
  - 5.1.1 The QK Critical Section 28
  - 5.1.2 The Thread-Local Storage for Newlib 28
- 5.2 The alt_hooks.h Header File 29
  - 5.2.1 The QK-Specific Interrupt Entry and Exit 30
- 5.3 The Heap and Environment Mutexes 30
- 5.4 Idle Loop Customization in QK_onIdle() 30

6 QS Software Tracing 32

7 Related Documents and References 35

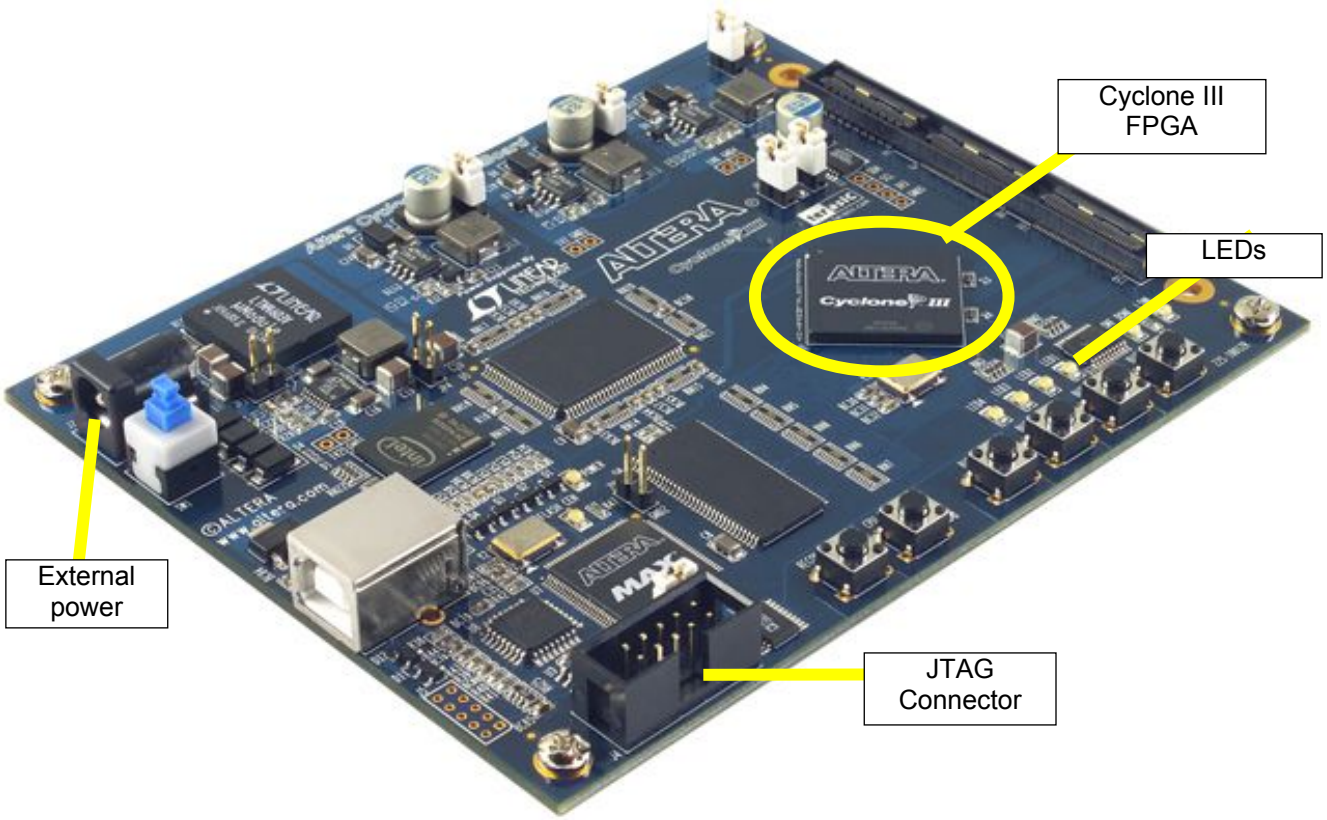
8 Contact Information 36

Copyright © Quantum Leaps, LLC. All Rights Reserved.

1 Introduction

This **QP™ Development Kit** (QDK) describes how to use QP™ event-driven platform with the Altera® Nios II IDE toolset. The actual hardware/software used to test this QDK as shown in Figure 1 and described below:

Figure 1 Altera Nios II Embedded Evaluation Kit (NEEK), Cyclone III edition [Altera 09]).

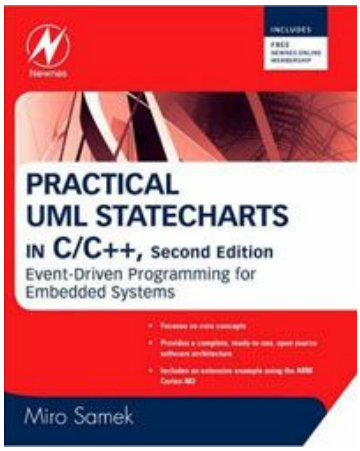


1. Altera Nios II Embedded Evaluation Kit (NEEK), Cyclone III edition
2. Altera Nios II IDE version 9.1 (based on Eclipse and the GNU toolchain)
3. Altera Quartus II version 9.1 design tool
4. QP/C/C++ version **4.5.03** or higher.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

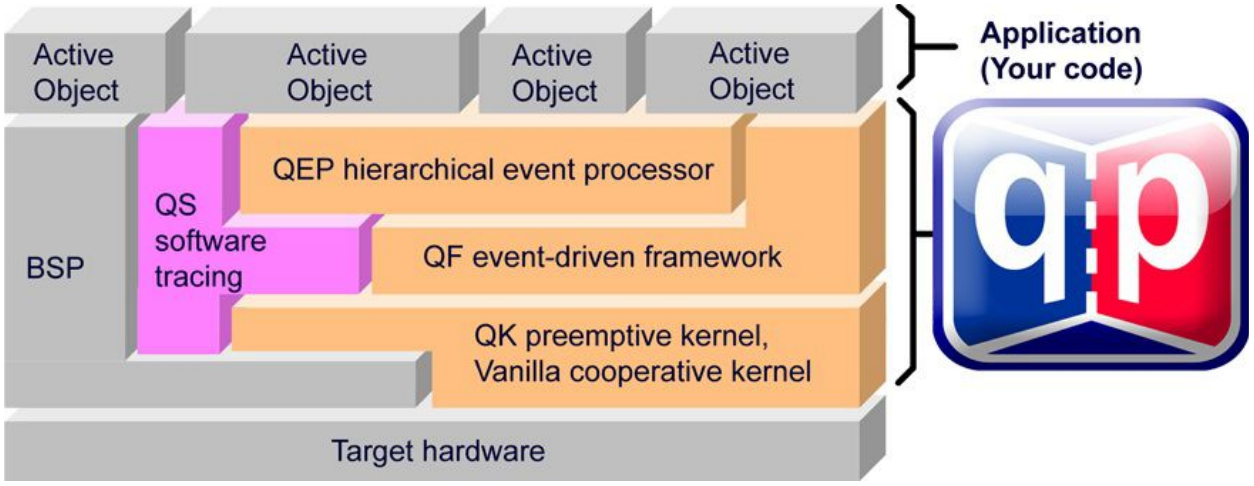
1.1 About QP™

QP™ is a family of very lightweight, open source, state machine-based frameworks for developing event-driven applications. QP enables building well-structured embedded applications as a set of concurrently executing hierarchical state machines (UML statecharts) directly in C or C++ **without big tools**. QP is described in great detail in the book "*Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems*" [PSiCC2] (Newnes, 2008).



As shown in Figure 2, QP consists of a universal UML-compliant event processor (QEP), a portable real-time framework (QF), a tiny run-to-completion kernel (QK), and software tracing instrumentation (QS). Current versions of QP include: QP/C™ and QP/C++™, which require about 4KB of code and a few hundred bytes of RAM, and the ultra-lightweight QP-nano, which requires only 1-2KB of code and just several bytes of RAM. The Linux port described in this Application Note pertains to QP/C and QP/C++.

Figure 2 QP components and their relationship with the target hardware, board support package (BSP), and the application



QP can work with or without a traditional RTOS or OS. In the simplest configuration, QP can completely **replace** a traditional RTOS. QP can manage up to 63 concurrently executing tasks structured as state machines (called active objects in UML).

1.2 What's Included in the QDK-Nios-II?

This QDK provides the QP port to Nios II with the Altera Nios II IDE and two example applications described in the Quantum Leaps Application Note "Dining Philosophers Example" [QL AN-DPP 08]:

1. Dining Philosopher Problem for the QF "vanilla" port; and
2. Dining Philosopher Problem for the QK preemptive kernel.

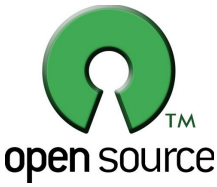
Copyright © Quantum Leaps, LLC. All Rights Reserved.

Both examples demonstrate the QS ("Spy") instrumentation.

NOTE: This QDK covers both the C and C++ version of QP. The concrete code examples are taken from the C version, but the C++ version is essentially identical except for some trivial syntax differences.

1.3 Licensing QP™

The **Generally Available (GA)** distribution of QP™ available for download from the www.state-machine.com/downloads website is offered with the following two licensing options:



- The GNU General Public License version 2 (GPL) as published by the Free Software Foundation and appearing in the file `GPL.TXT` included in the packaging of every Quantum Leaps software distribution. The GPL *open source* license allows you to use the software at no charge under the condition that if you redistribute the original software or applications derived from it, the complete source code for your application must be also available under the conditions of the GPL (GPL Section 2[b]).

- One of several Quantum Leaps commercial licenses, which are designed for customers who wish to retain the proprietary status of their code and therefore cannot use the GNU General Public License. The customers who license Quantum Leaps software under the commercial licenses do not use the software under the GPL and therefore are not subject to any of its terms.

For more information, please visit the licensing section of our website at: www.state-machine.com/licensing.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

2 Installation

This section describes how to install, build, and use QDK-AVR-xmega-IAR based two examples. This information is intentionally included early in this document, so that you could start using the QDK as soon as possible. The main focus of this section is to walk you quickly through the main points without slowing you down with full-blown detail.

> **NOTE:** This QDK assumes that the standard QP distribution consisting of QEP, QF, and QK has been installed, before installing this QDK. It is also strongly recommended that you read the QP Tutorial (www.quantum-leaps.com/doxygen/qpc/tutorial_page.html) before you start experimenting with this QDK.

The QDK code is distributed in a ZIP archive (`qdkc_nios2-gnu_<ver>.zip`, where `<ver>` stands for a specific QDK-Nios2-GNU version, such as 4.5.03). You can uncompress the archive into the same directory into which you've installed all the standard QP components. The installation directory you choose will be referred henceforth as QP Root Directory `<qp>`. The following Listing 1 shows the directory structure and selected files included in the QDK-Nios2-GNU distribution. (Please note that the QP directory structure is described in detail in a separate Application Note: "QP Directory Structure":

Listing 1 Selected QP directories and files after installing QDK-Nios2-GNU. The highlighted elements are included in the standard QDK-Nios2-GNU distribution.

```
<qpc>\                      - QP/C root directory (qpcpp for QP/C++)
  |
  +-include\                - QP public include files
  | +-qassert.h             - Quantum Assertions platform-independent public include
  | +-qevt.h                - QEvt include
  | +-qep.h                 - QEP platform-independent public include
  | +-qf.h                  - QF platform-independent public include
  | +-qk.h                  - QK platform-independent public include
  | +-qequeue.h             - native QF event queue include
  | +-qmpool.h              - native QF memory pool include
  | +-qpset.h               - native QF priority set include
  |
  +-ports\                  - QP ports
  | +-nios2\                - Nios2 port
  | | +-qk\                 - QK (Quantum Kernel) ports
  | | | +-gnu\              - examples compiled with the GNU toolset
  | | | | +-os\             - Header files overriding the standard Altera HAL
  | | | | | +-alt_flag.h        - Header overriding the standard HAL flag setting
  | | | | | +-alt_hooks.h       - Header overriding the standard HAL hooks
  | | | | | +-alt_sem.h         - Header overriding the standard HAL semaphores
  | | | | | +-alt_env_lock.c    - Implementation of environment locks
  | | | | | +-alt_malloc_lock.c - Implementation of malloc locks
  | | | | +-qep_port.h      - QEP port to Nios II
  | | | | +-qf_port.h       - QF port to Nios II
  | | | | +-qk_port.h       - QK port to Nios II
  | | | | +-qk_port.c       - QK port to Nios II (source)
  | | | | +-qs_port.h       - QS port to Nios II
  | | | | +-qp_port.h       - QP port to Nios II
  | | | |
  | | +-vanilla\            - "vanilla" ports
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
| | | +-gnu\              - examples compiled with the GNU toolset
| | | | +-os\             - Header files overriding the standard Altera HAL
| | | | | +-alt_hooks.h   - Header overriding the standard HAL hooks
| | | | +-qep_port.h      - QEP port to Nios II
| | | | +-qf_port.h       - QF port to Nios II
| | | | +-qF_port.c       - QF port to Nios II (source)
| | | | +-qs_port.h       - QS port to Nios II
| | | | +-qp_port.h       - QP port to Nios II
|
+-examples\                - subdirectory containing the examples
| +-nios2\                 - Nios2 port
| | +-qk\                  - QK (Quantum Kernel) examples
| | | +-gnu\               - examples compiled with the GNU toolset
| | | | +-dpp-qk_cycloneIII_3c25_niosII\ - DPP example for NEEK CycloneIII
| | | | | +-Debug\       - directory containing the Debug build
| | | | | +-Release\     - directory containing the Release build
| | | | | +-Spy\         - directory containing the Spy build
| | | | | +-bsp.c        - Board Support Package for AVR-xmega
| | | | | +-bsp.h        - BSP header file
| | | | | +-main.c       - the main function
| | | | | +-philo.c      - the Philosopher active object
| | | | | +-dpp.h        - the DPP header file
| | | | | +-table.c      - the Table active object
| | | |
| +-nios2\                 - Nios2 port
| | +-qk\                  - "Vanilla" Kernel examples
| | | +-gnu\               - examples compiled with the GNU toolset
| | | | +-dpp_cycloneIII_3c25_niosII\ - DPP example for NEEK CycloneIII
| | | | | +-Debug\       - directory containing the Debug build
| | | | | +-Release\     - directory containing the Release build
| | | | | +-Spy\         - directory containing the Spy build
| | | | | +-bsp.c        - Board Support Package for AVR-xmega
| | | | | +-bsp.h        - BSP header file
| | | | | +-main.c       - the main function
| | | | | +-philo.c      - the Philosopher active object
| | | | | +-dpp.h        - the DPP header file
| | | | | +-table.c      - the Table active object
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

3 Creating Example Application and BSP

This section explains step by step the process of creating the example application and the Board Support Package (BSP) with the Eclipse-based Altera Nios II IDE. The starting point for this project is the "Hello World" template that uses the Altera HAL. This will create a BSP and an application that will run on the NEEK and output to the Nios II Console. We add to this project a Nios II Library that contains QP. Then modify the BSP settings and the Application settings to include and link with the QP Library.

NOTE: The Nios II IDE is based on the Eclipse CDT (C/C++ Development Tooling). The CDT workspaces and project files are notoriously difficult to move from one development workstation to another because they contain absolute paths. Therefore, instead of providing ready-to-use projects, this document describes how to generate the QP System Library and the application with the Nios II IDE. The additional advantage of this approach is that you can follow the same steps to generate the QP System Library for different hardware designs and/or different FPGAs.

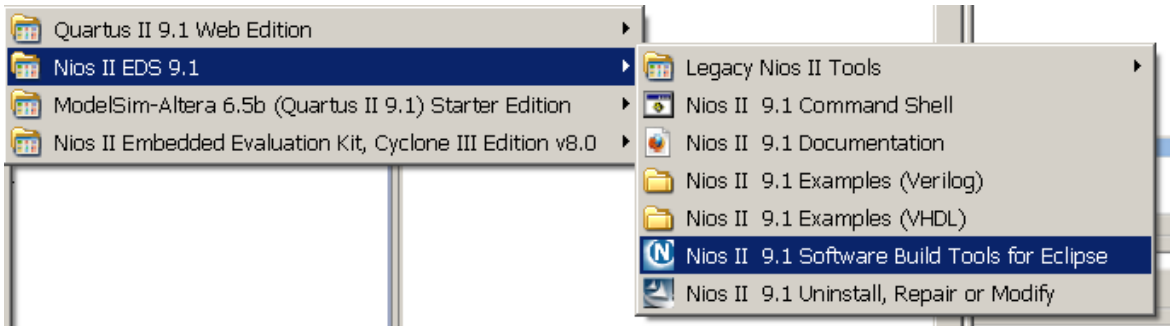
This section is based on the following resources:

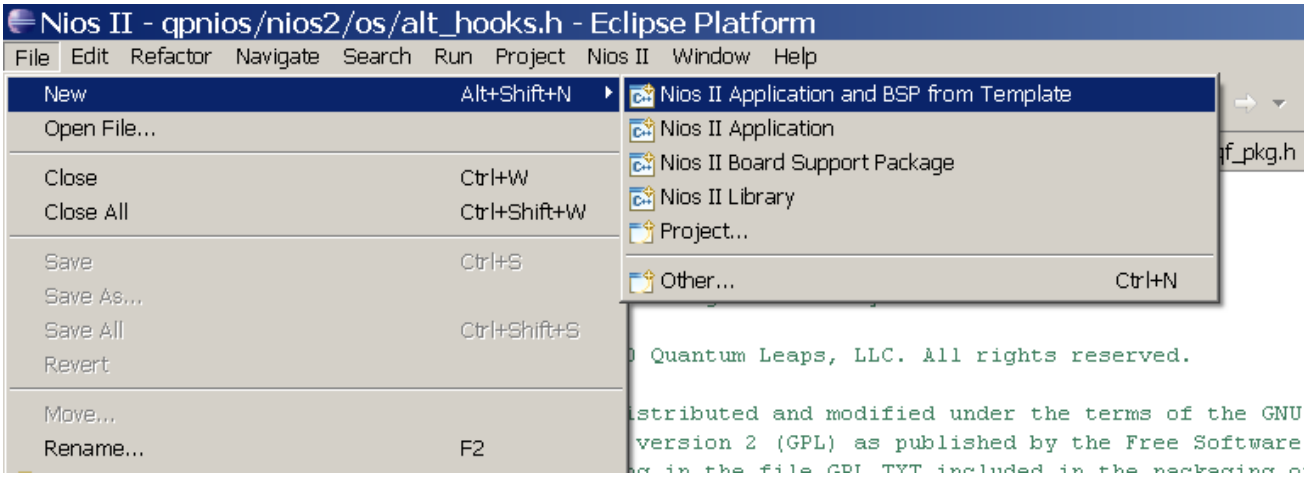
Standard Hardware Design, NEEK Cyclone III Info [NEEK 3C25]

Hardware Design ZIP file

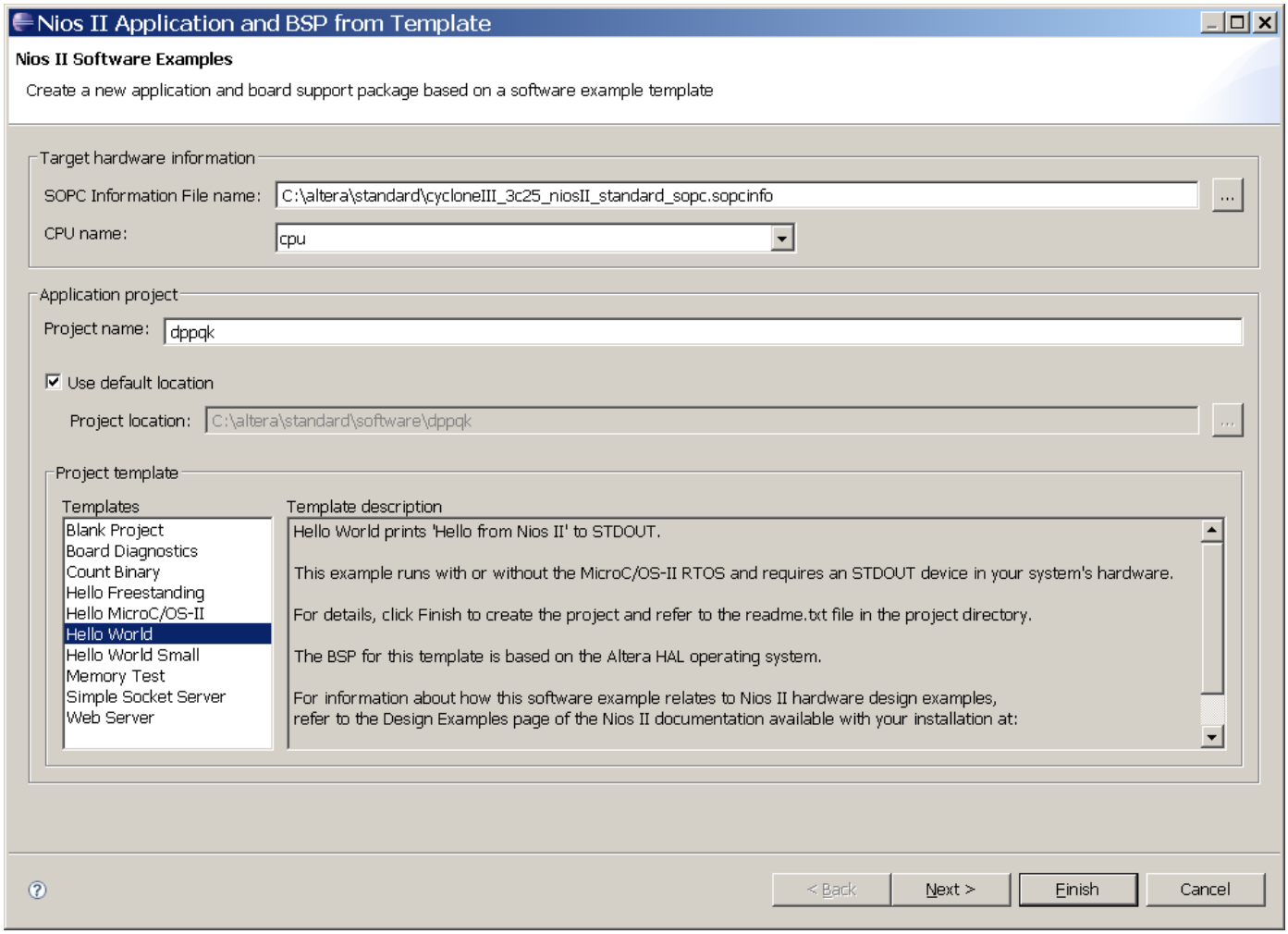
3.1 Launching the Nios II IDE

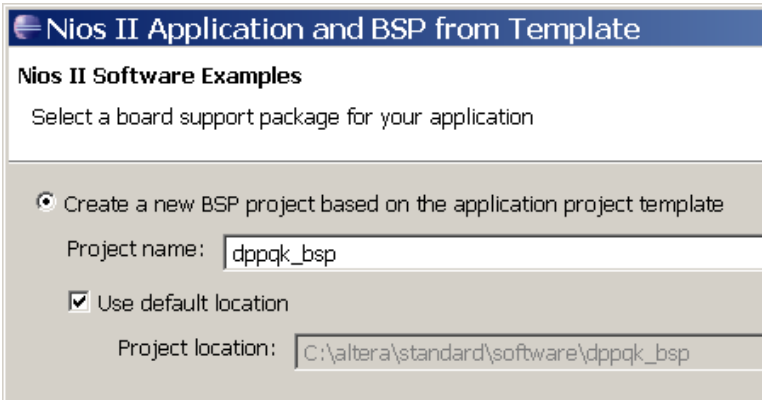
Launch Nios II IDE from the Atera submenu from the Start menu.


Quartus II 9.1 Web Edition
Nios II EDS 9.1
ModelSim-Altera 6.5b (Quartus II 9.1) Starter Edition
Nios II Embedded Evaluation Kit, Cyclone III Edition v8.0
Legacy Nios II Tools
Nios II 9.1 Command Shell
Nios II 9.1 Documentation
Nios II 9.1 Examples (Verilog)
Nios II 9.1 Examples (VHDL)
Nios II 9.1 Software Build Tools for Eclipse
Nios II 9.1 Uninstall, Repair or Modify


Nios II - qpnios/nios2/os/alt_hooks.h - Eclipse Platform
File Edit Refactor Navigate Search Run Project Nios II Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Nios II Application and BSP from Template
Nios II Application
Nios II Board Support Package
Nios II Library
Project...
Other... Ctrl+N
Quantum Leaps, LLC. All rights reserved.
istributed and modified under the terms of the GNU
version 2 (GPL) as published by the Free Software

Copyright © Quantum Leaps, LLC. All Rights Reserved.

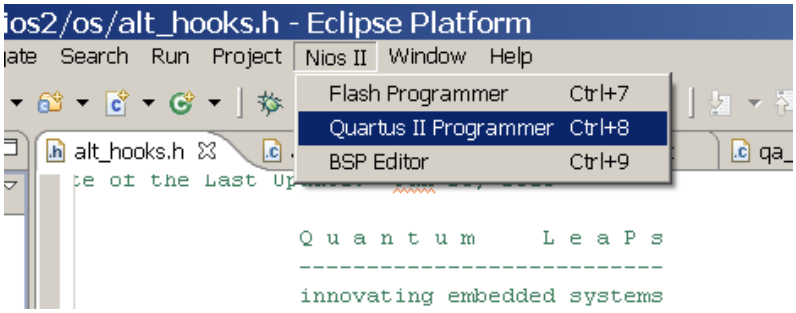

Nios II Application and BSP from Template
Nios II Software Examples
Create a new application and board support package based on a software example template
Target hardware information
SOPC Information File name:
C:\altera\standard\cycloneIII_3c25_niosII_standard_sopc.sopcinfo
CPU name:
cpu
Application project
Project name:
dppqk
Use default location
Project location:
C:\altera\standard\software\dppqk
Project template
Templates
Blank Project
Board Diagnostics
Count Binary
Hello Freestanding
Hello MicroC/OS-II
Hello World
Hello World Small
Memory Test
Simple Socket Server
Web Server
Template description
Hello World prints 'Hello from Nios II' to STDOUT.
This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.
For details, click Finish to create the project and refer to the readme.txt file in the project directory.
The BSP for this template is based on the Altera HAL operating system.
For information about how this software example relates to Nios II hardware design examples,
refer to the Design Examples page of the Nios II documentation available with your installation at:
< Back
Next >
Finish
Cancel


Nios II Application and BSP from Template
Nios II Software Examples
Select a board support package for your application
Create a new BSP project based on the application project template
Project name:
dppqk_bsp
Use default location
Project location:
C:\altera\standard\software\dppqk_bsp

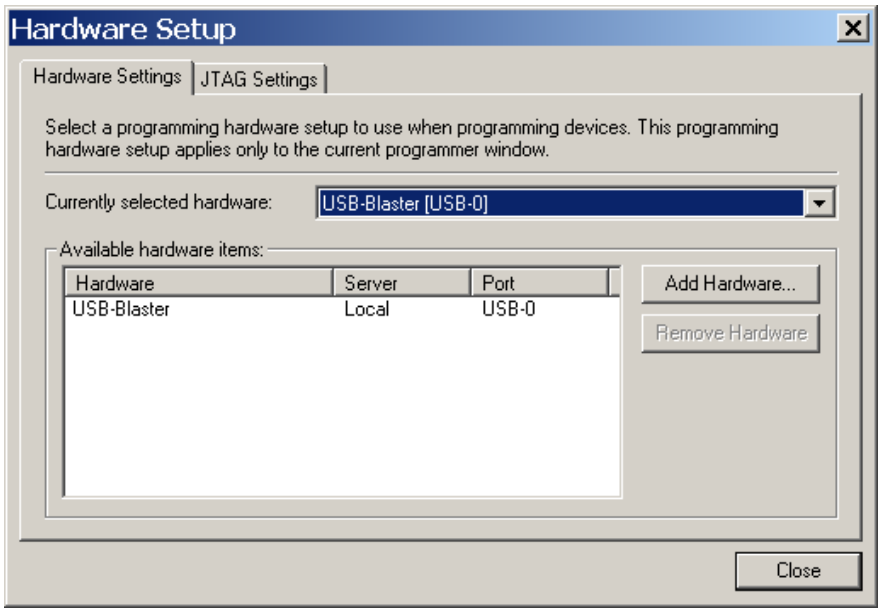
Copyright © Quantum Leaps, LLC. All Rights Reserved.

3.2 Configuring the FPGA

- Connect NEEK AC adaptor.
- Connect JTAG USB cable.
- Power on the NEEK.
- Launch Quartus II Programmer.

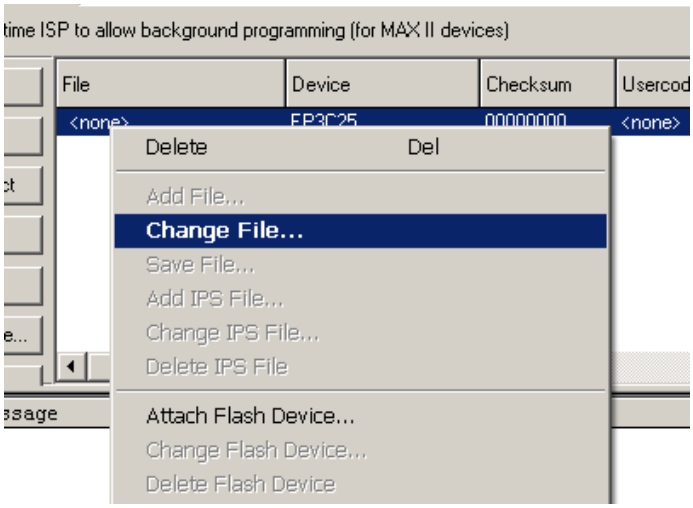

ios2/os/alt_hooks.h - Eclipse Platform
ate Search Run Project Nios II Window Help
Flash Programmer Ctrl+7
Quartus II Programmer Ctrl+8
BSP Editor Ctrl+9
alt_hooks.h
qa_
Q u a n t u m L e a P s
innovating embedded systems

- Perform Hardware Setup.

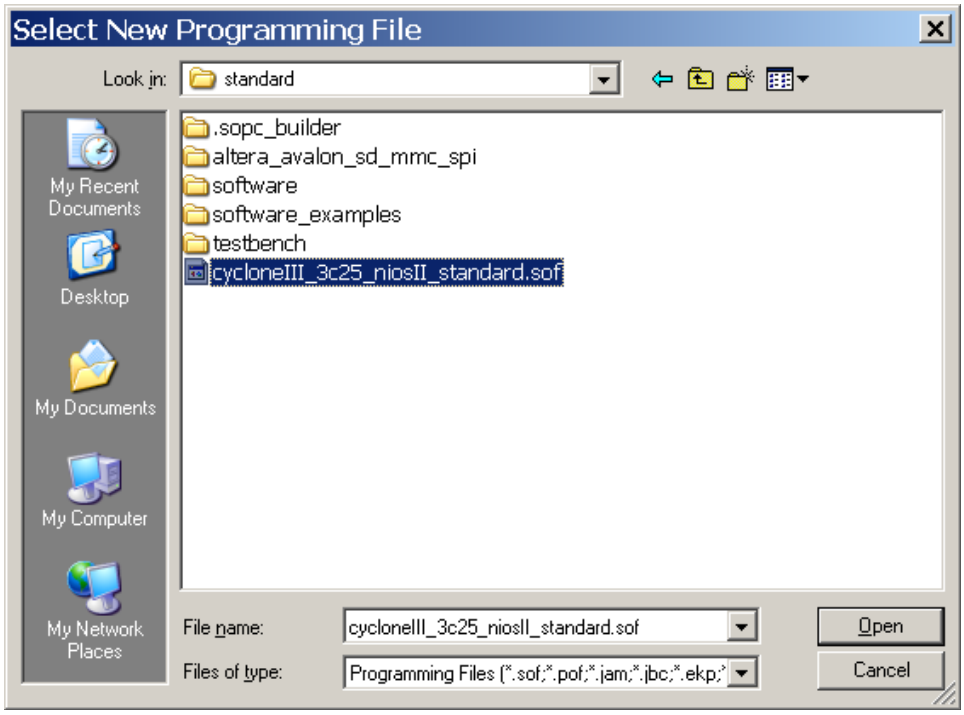


- Press Auto-Detect.
- Right click on file and select Change file.

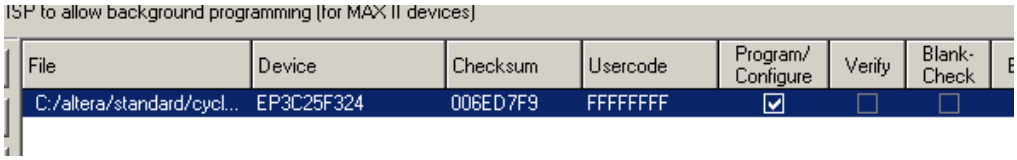
Copyright © Quantum Leaps, LLC. All Rights Reserved.



- Select the `.sof` file matching the `sopcinfo` file used to create the BSP.

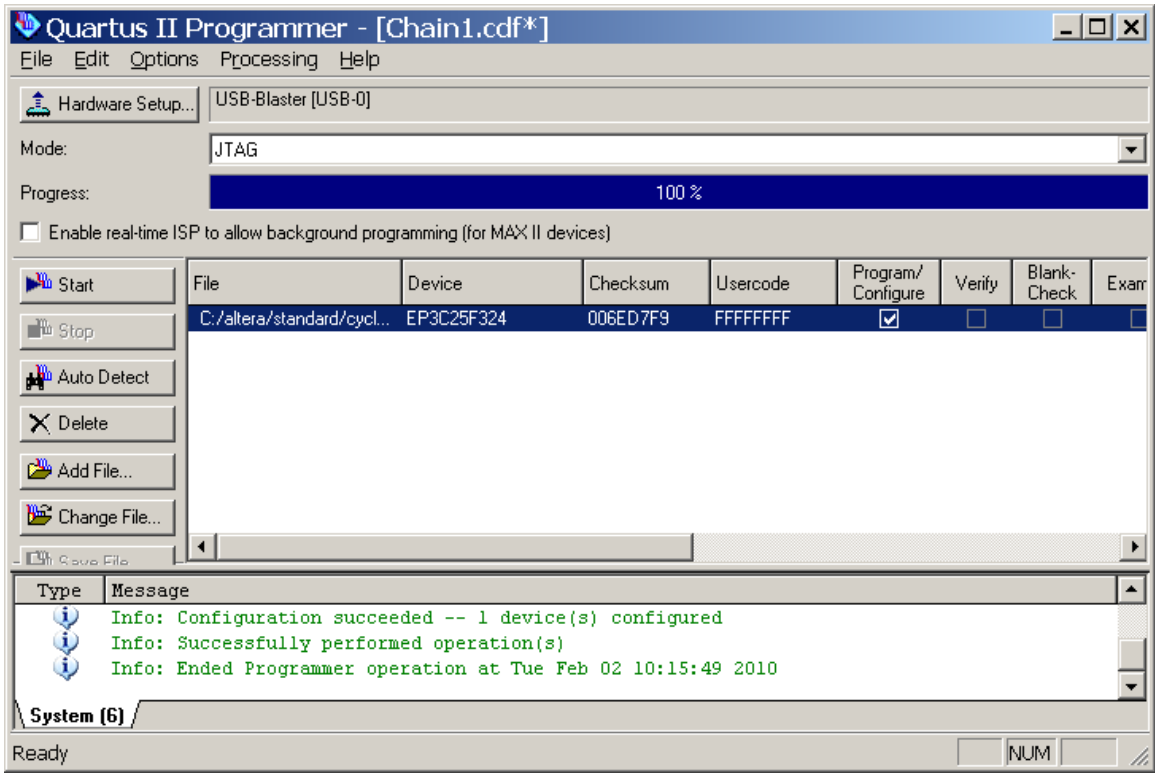


- Check the Program/Configure check box.



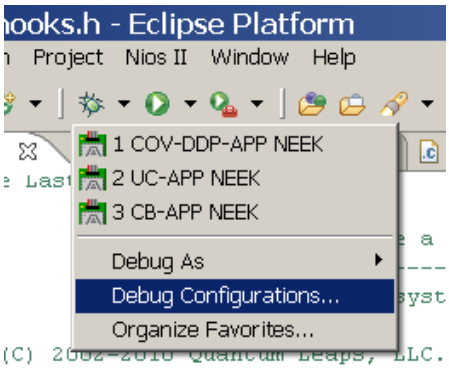
- Click Start. This will download the FPGA configuration to the NEEK.

Copyright © Quantum Leaps, LLC. All Rights Reserved.



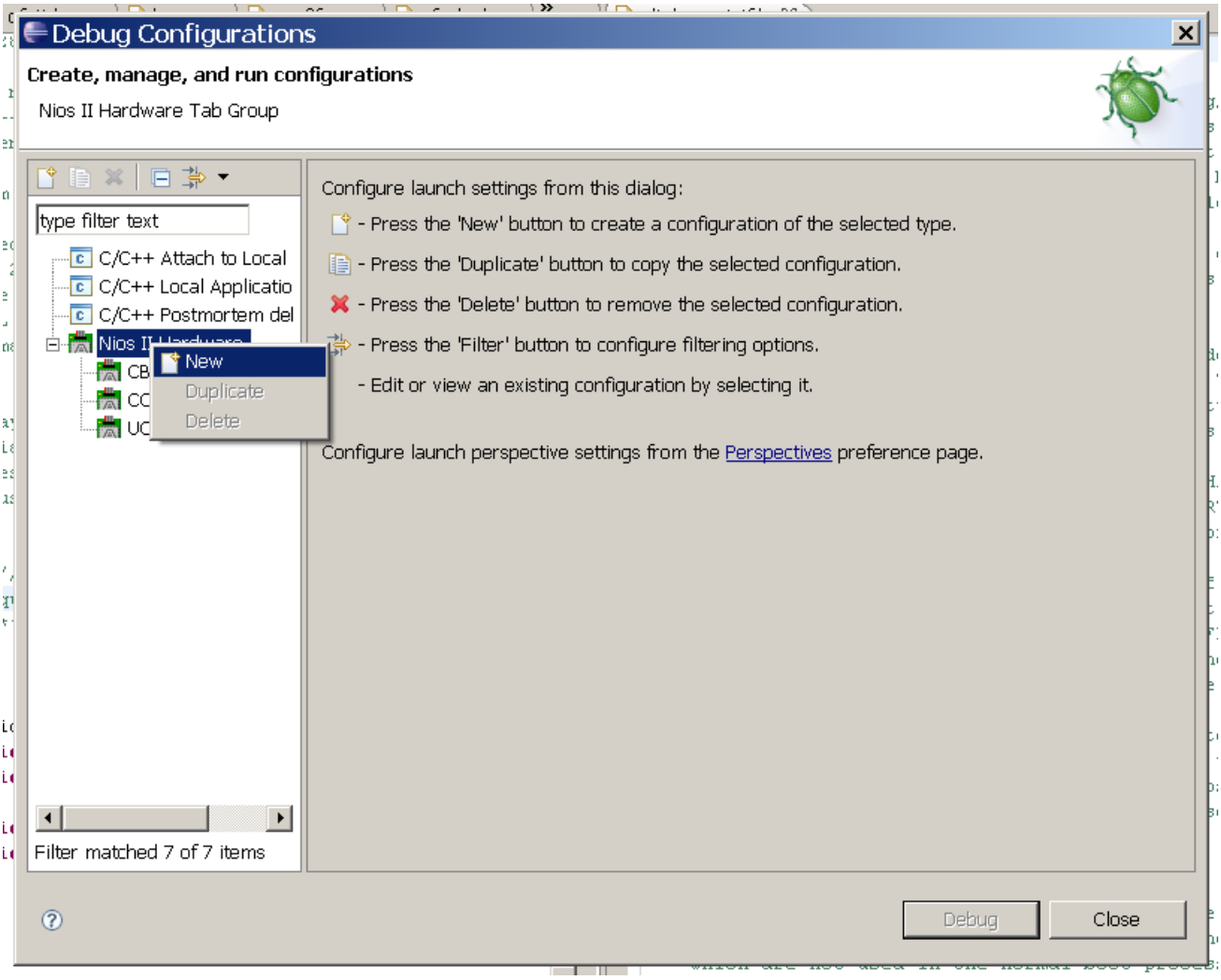
3.3 Debugging the Application

- Bring up Debug Configuration window:

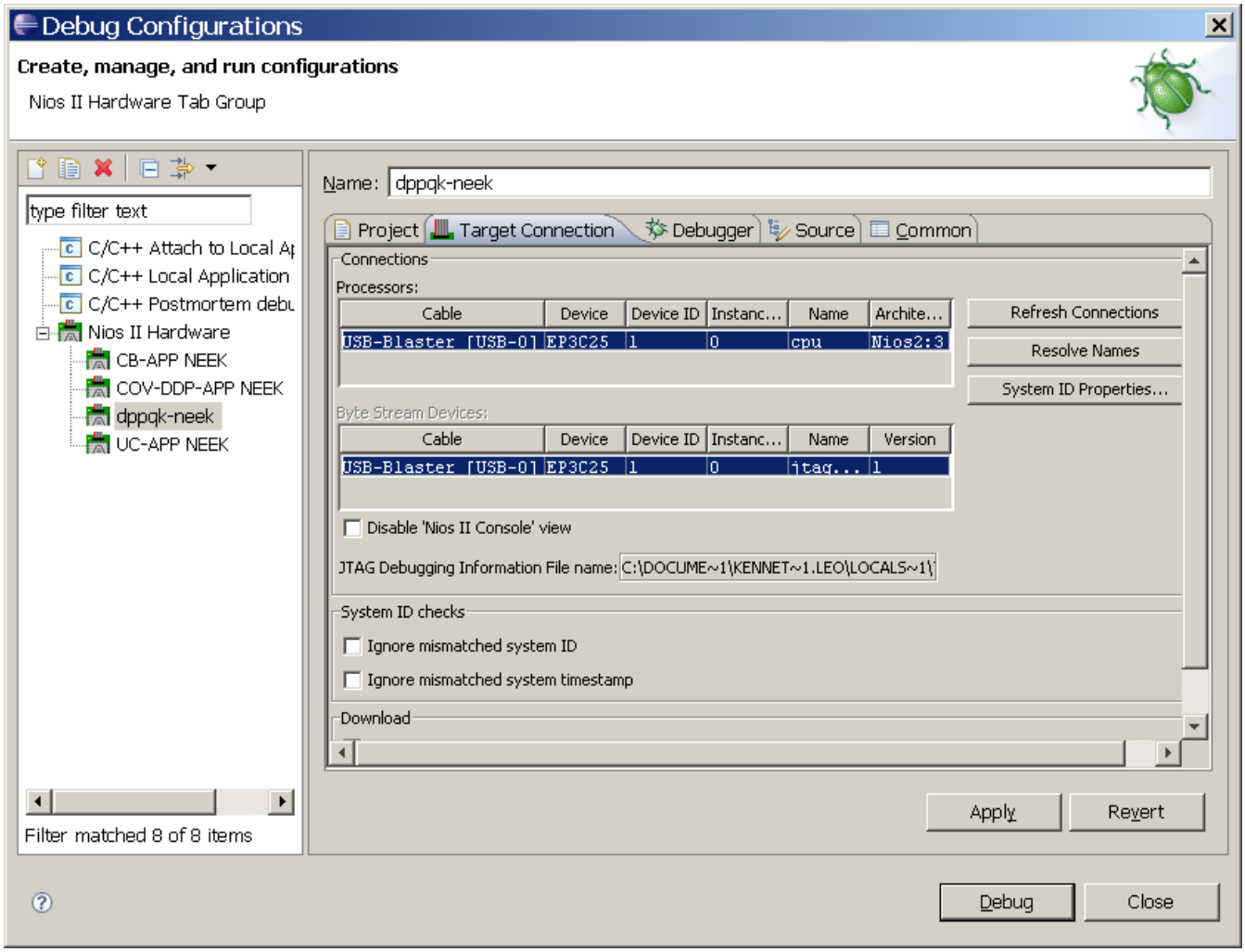


- Create new Nios II Hardware debug configuration.

Copyright © Quantum Leaps, LLC. All Rights Reserved.


Debug Configurations
Create, manage, and run configurations
Nios II Hardware Tab Group
type filter text
C/C++ Attach to Local
C/C++ Local Applicatio
C/C++ Postmortem del
New
Duplicate
Delete
Filter matched 7 of 7 items
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
Configure launch perspective settings from the Perspectives preference page.
Debug
Close

Copyright © Quantum Leaps, LLC. All Rights Reserved.

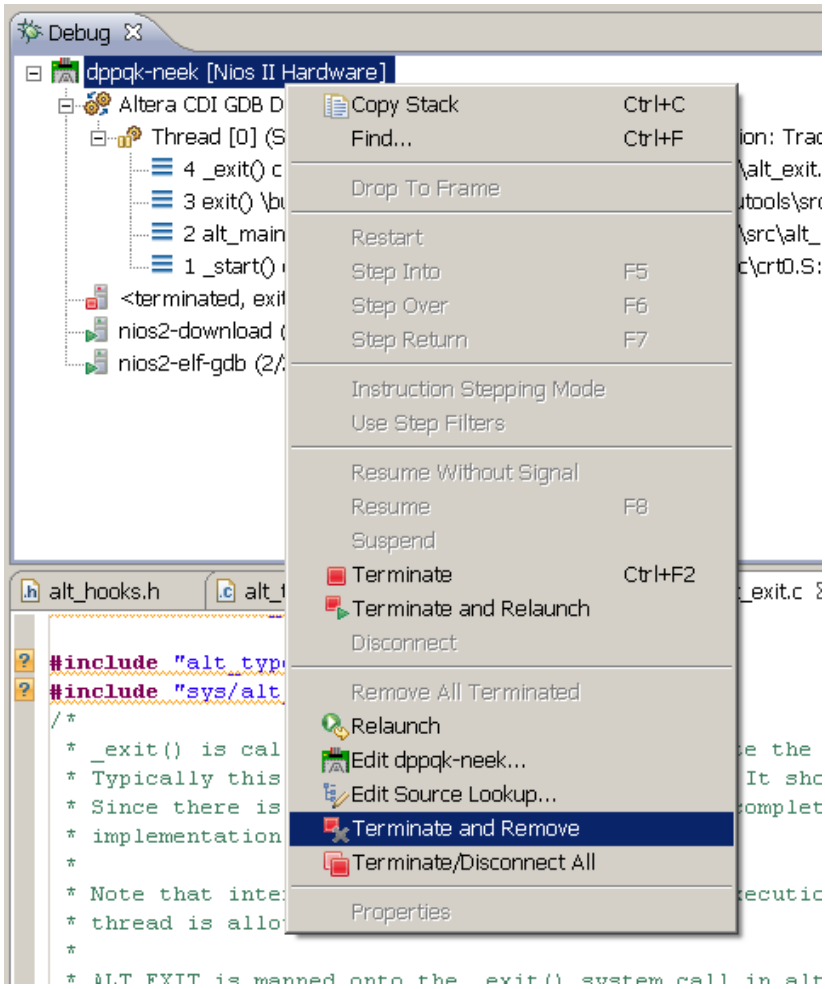


- Name the configuration.
- Go to the Target Connection tab and click Refresh Connections.
- Click Debug.

NiosII IDE will download to the target, and break at main.

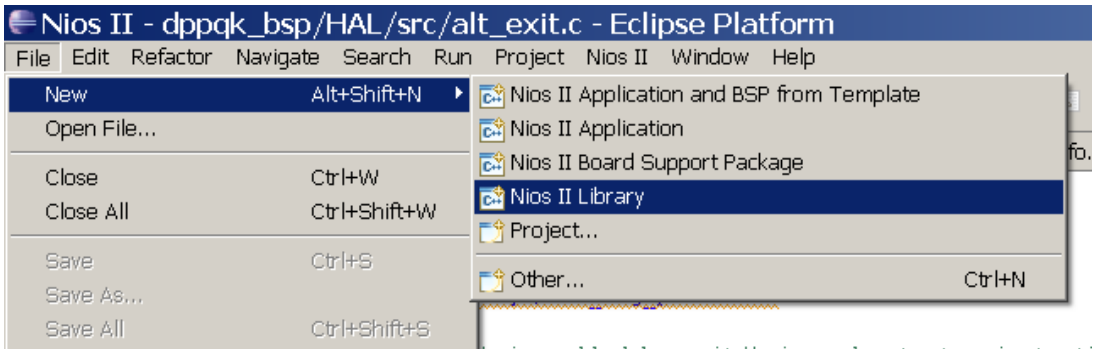
- Click resume to continue debugging.
- Since an exit from main doesn't really exit (see alt_exit.c) you have to kill the debug run manually:

Copyright © Quantum Leaps, LLC. All Rights Reserved.



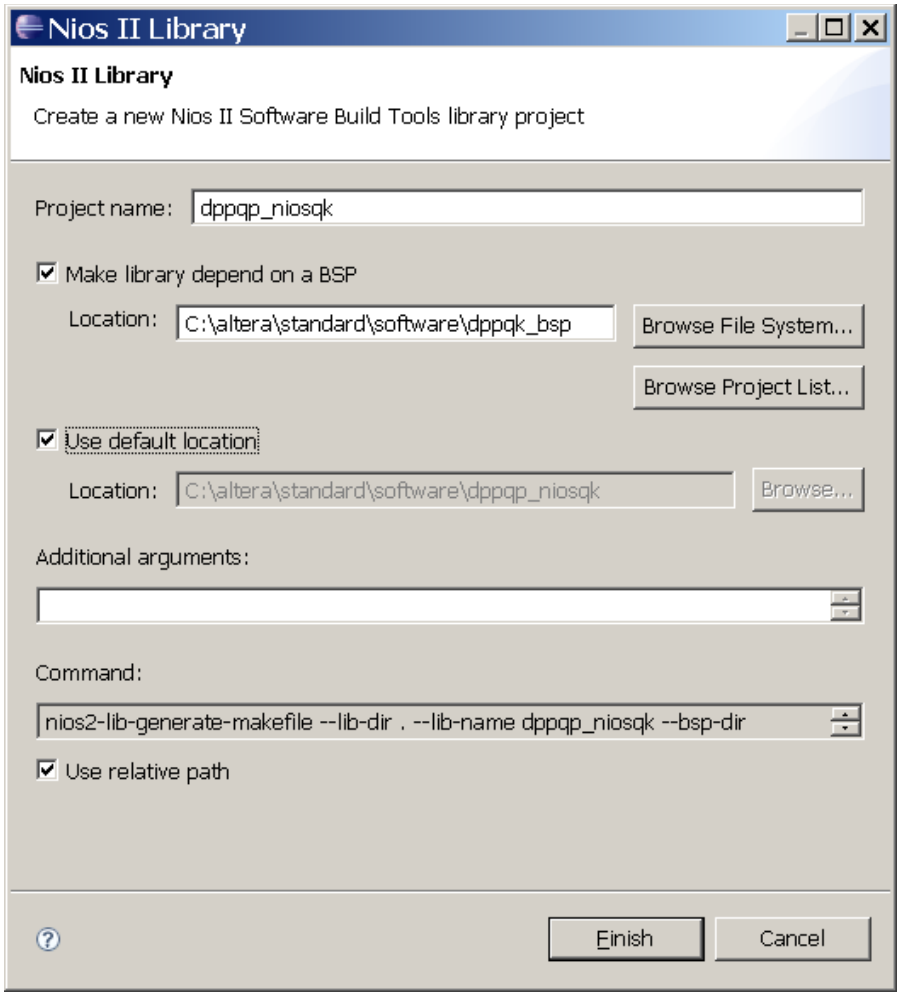
3.4 Creating the QP libraries (QEP, QF, QK, and QS)

- Create a library project from the template:

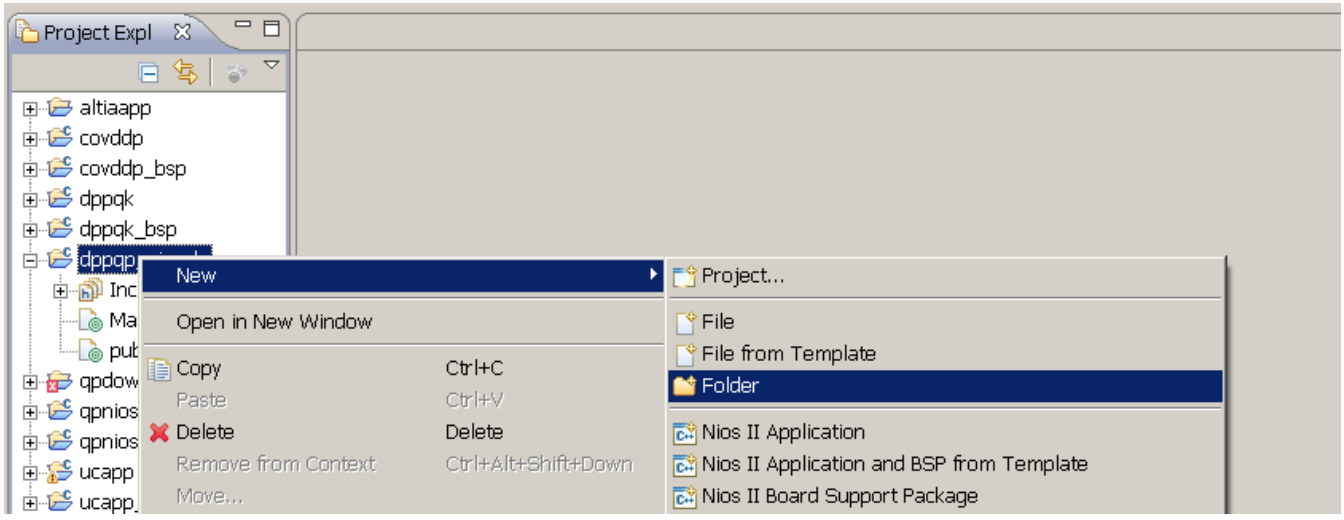


- Fill in the project details, and point the library to the previous created BSP:

Copyright © Quantum Leaps, LLC. All Rights Reserved.

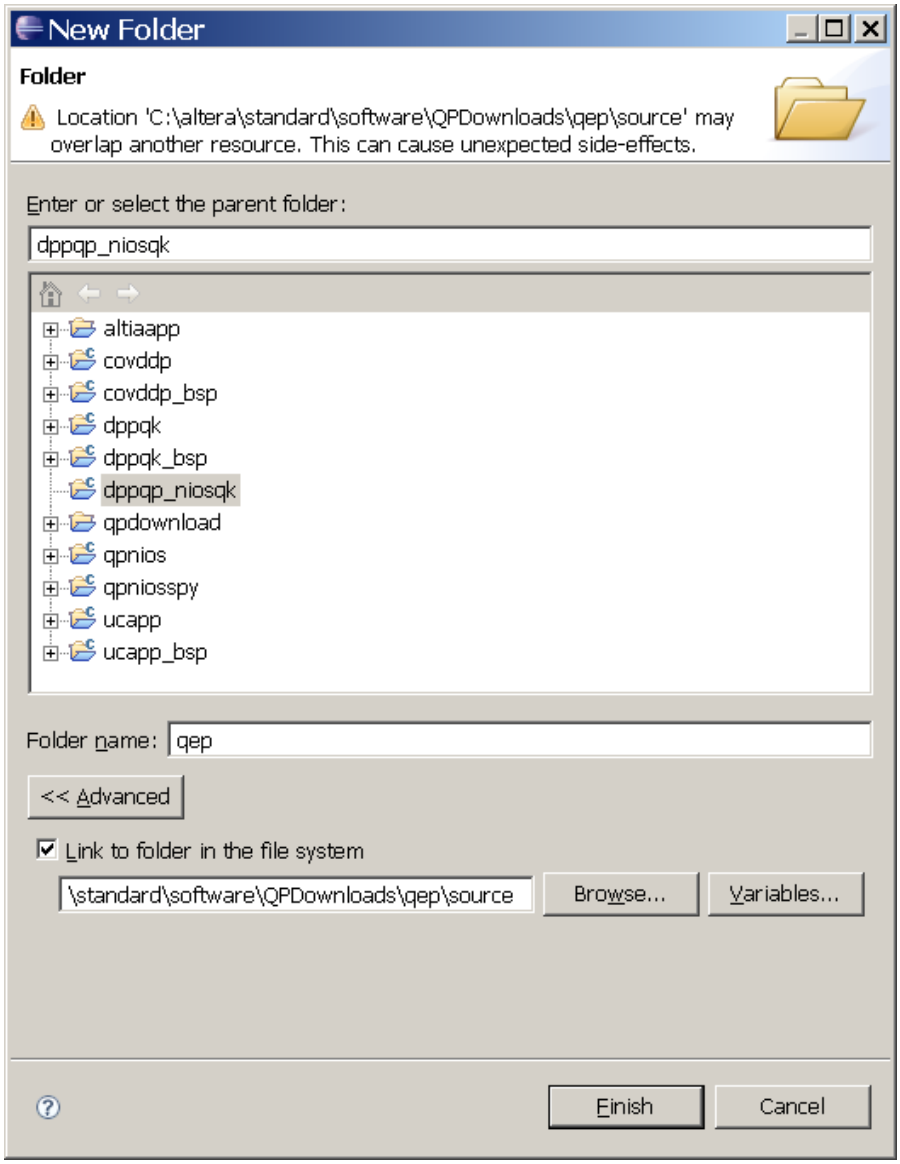


- Add a folder to the project for QEP



Copyright © Quantum Leaps, LLC. All Rights Reserved.

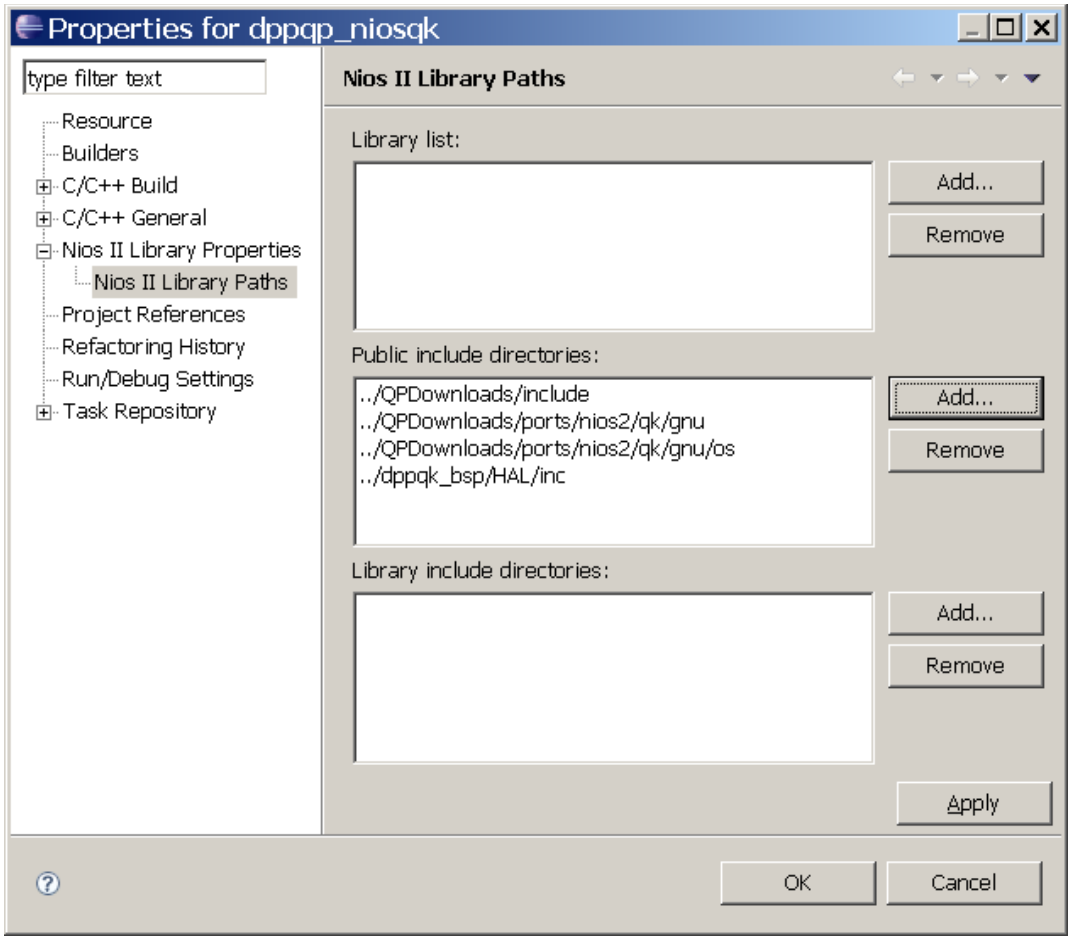
- Link the folder to the location of qep/source where QP was installed.



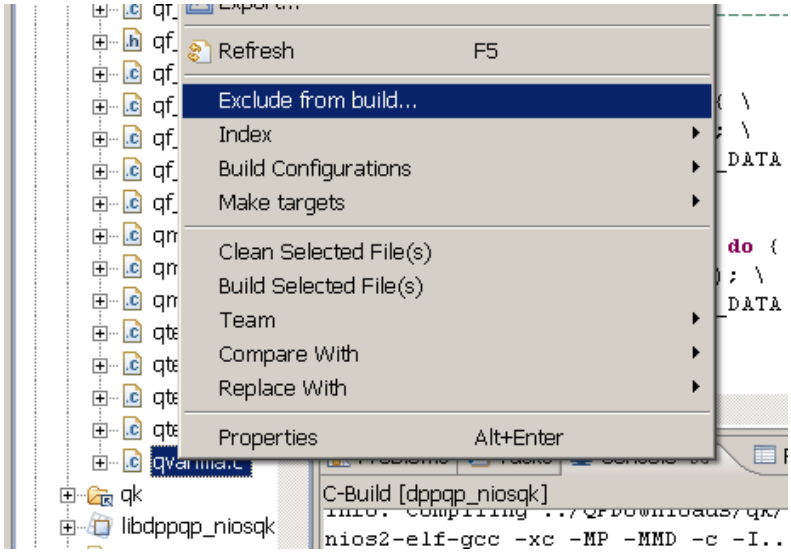
- Do the same for QF, QK, QS and the Nios II port files.

- In the project properties dialog, add the following paths:

Copyright © Quantum Leaps, LLC. All Rights Reserved.



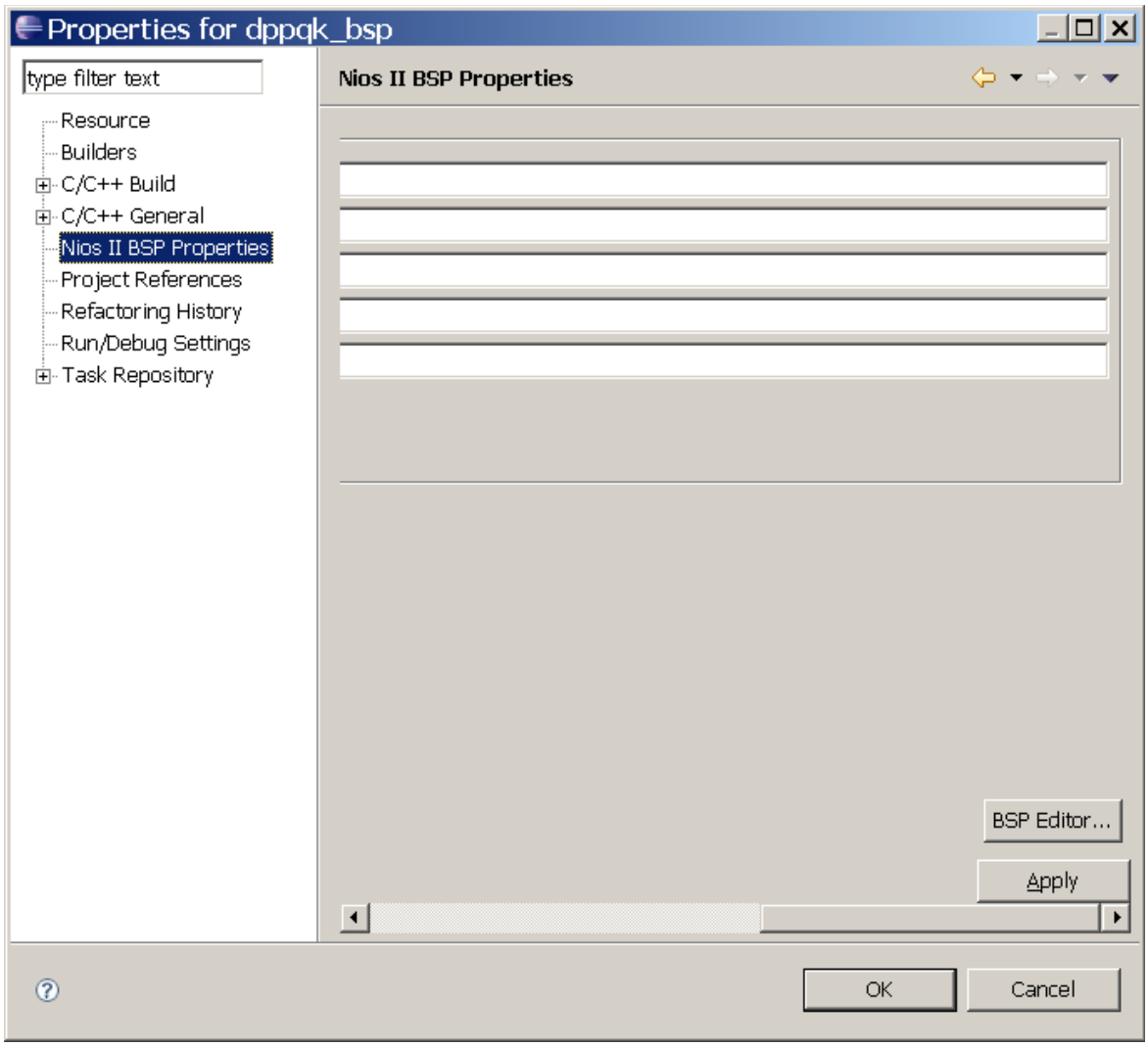
- Exclude `qf/scource/qvanilla.c` from the build. You may need to rename the file to prevent it from being built. The "Exclude from build…" didn't work for me for this file.



Copyright © Quantum Leaps, LLC. All Rights Reserved.

3.5 Configuring the BSP

- Bring up the properties of the BSP:



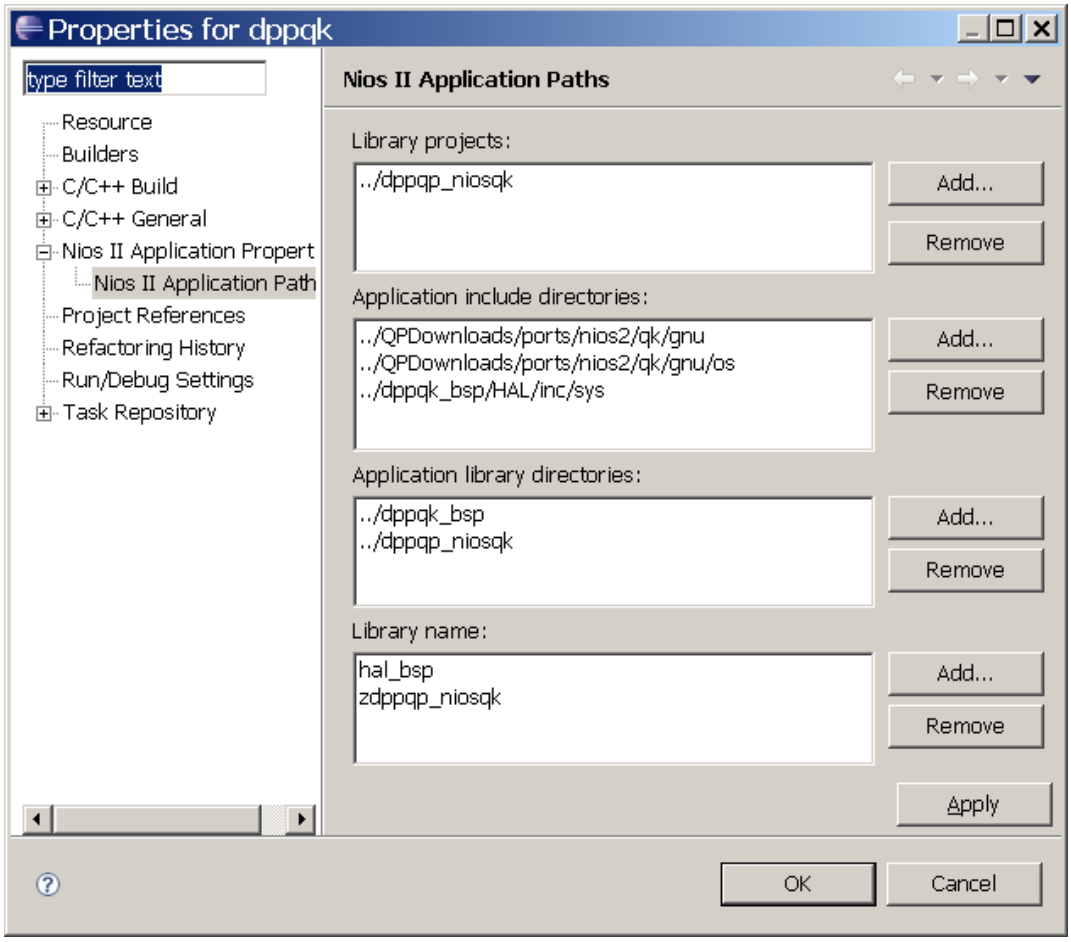
- Bring up the BSP editor.
- Add to Adanced->hal ->make->bsp_inc_dirs the paths to the port include directories. For example:
  - `<qp>\ports\nios2\qk\gnu`
  - `<qp>\ports\nios2\qk\gnu\os`

3.6 Configuring the Application

- Bring up the properties of the Application.
- Add includes paths, library paths.

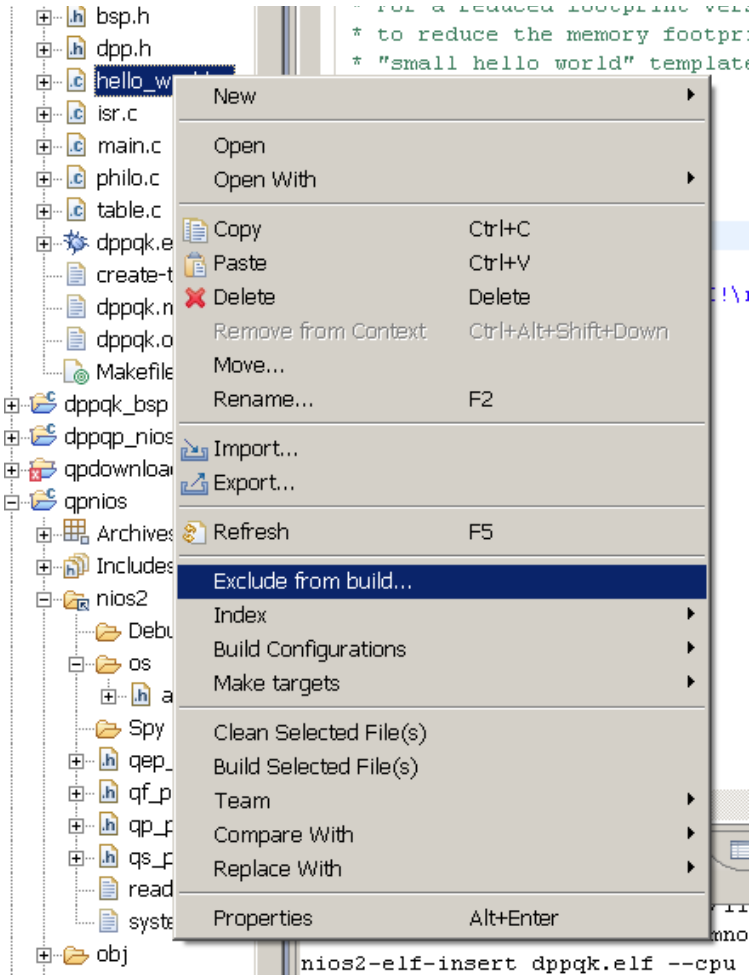
Copyright © Quantum Leaps, LLC. All Rights Reserved.

- Add libraries to the link phase. The Nios II Library Name field will sort alphabetically. Since BSP needs the QP Library to link properly, the QP Library must appear to the right of the BSP in the link line. The QP library was renamed to `zdppqp_niosqk` in this example to accomplish this.



- Copy the Dining Philosophers Problem (DPP) example code into the application project folder.
- Exclude `hello_world.c` from the build.

Copyright © Quantum Leaps, LLC. All Rights Reserved.



- Build and debug. Output to the Nios II Console will confirm that it is working.

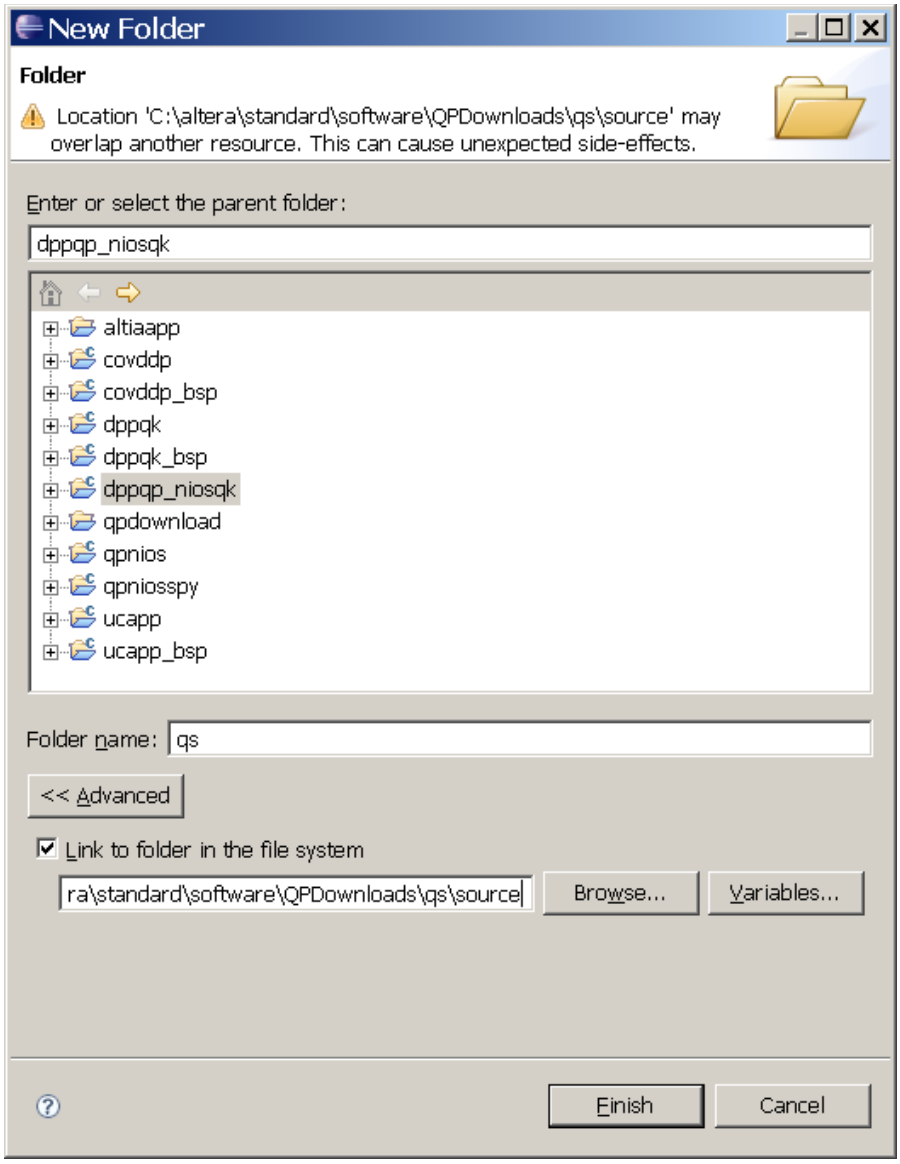
3.7 Using QS (Quantum Spy) Software Tracing

Unfortunately, the standard design for the NEEK does not have a UART. The JTAG UART is available though. But the QSPY host application cannot easily receive data from the JTAG UART. The `nios2-terminal` command line utility is used to capture the data coming over the JTAG UART. Redirect the output from `nios2-terminal` to a file. Then process the file using the QSPY host application (`-f<filename>` option).

3.7.1 Building the QS library

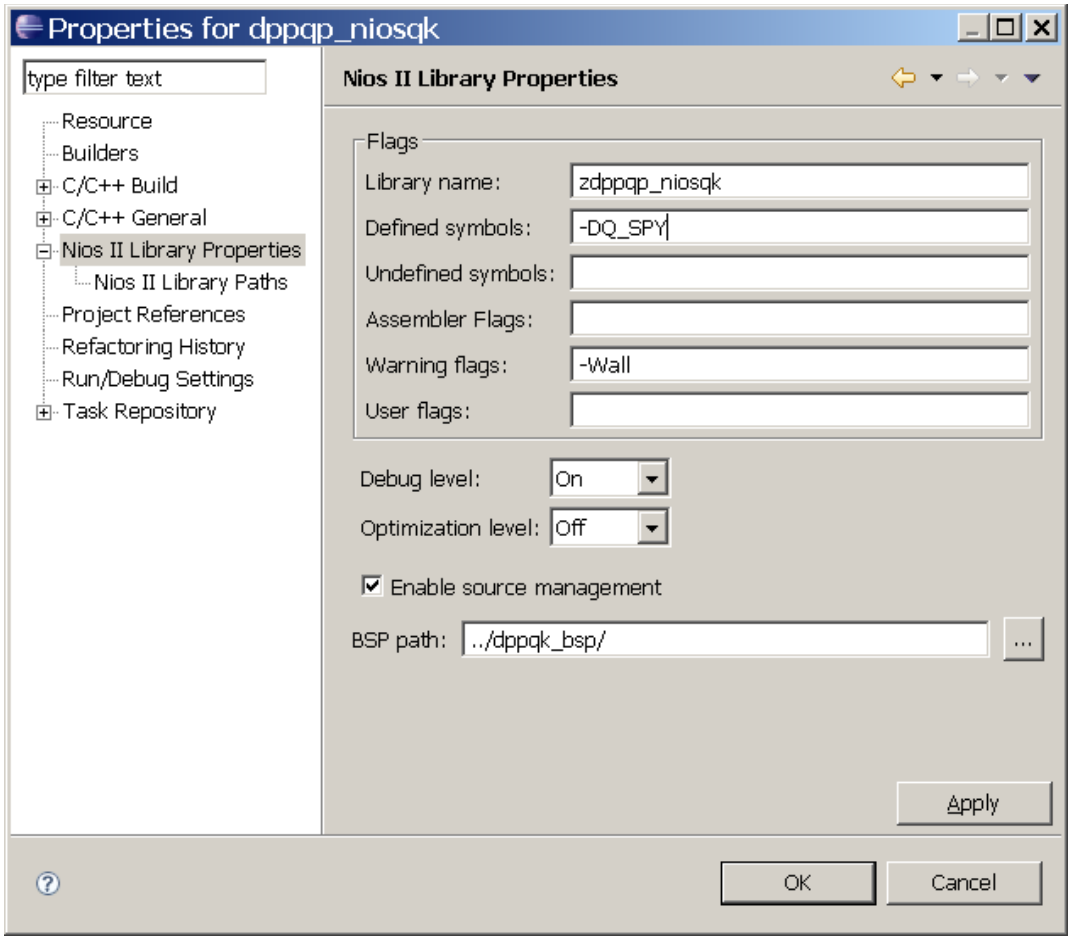
- Add a folder to the project for QS

Copyright © Quantum Leaps, LLC. All Rights Reserved.



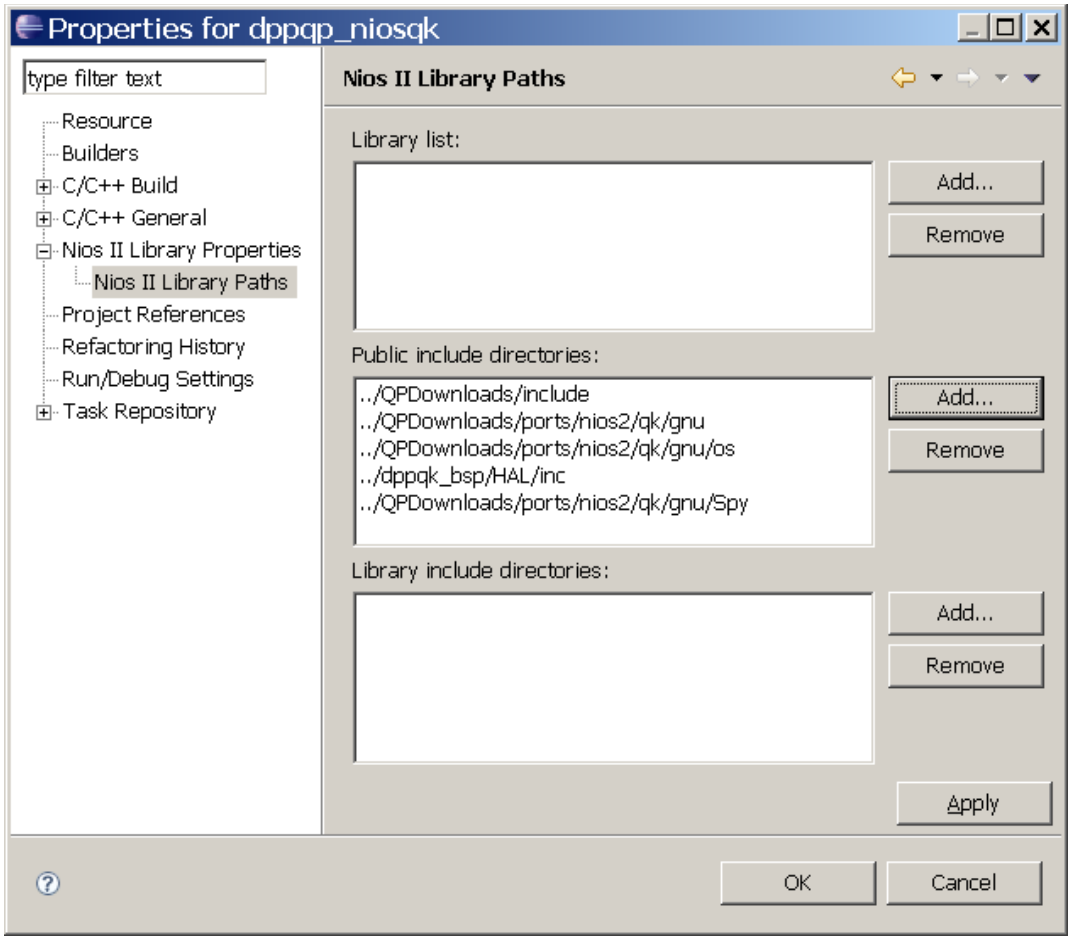
- Add the define `-DQ_SPY` to activate QS software tracing:

Copyright © Quantum Leaps, LLC. All Rights Reserved.



- Add the include path for QS
- In the Application project settings add the define `-DQ_SPY` to the application project settings as well.

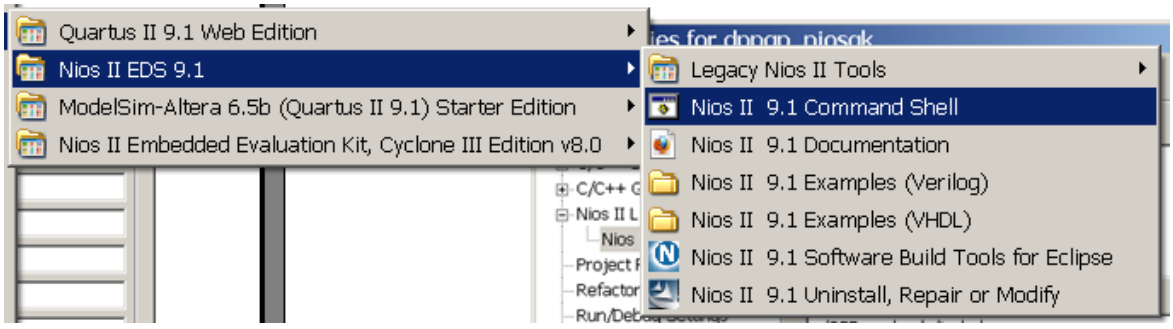
Copyright © Quantum Leaps, LLC. All Rights Reserved.



- Build.

3.7.2 *Collecting the QS software trace*

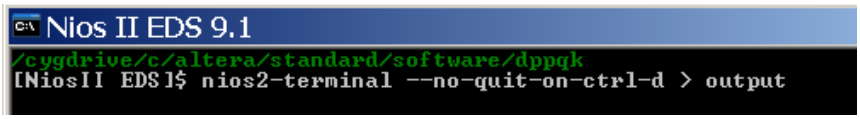
- Launch Command Shell twice (create two Command Shell windows):


Quartus II 9.1 Web Edition
Nios II EDS 9.1
ModelSim-Altera 6.5b (Quartus II 9.1) Starter Edition
Nios II Embedded Evaluation Kit, Cyclone III Edition v8.0
Legacy Nios II Tools
Nios II 9.1 Command Shell
Nios II 9.1 Documentation
Nios II 9.1 Examples (Verilog)
Nios II 9.1 Examples (VHDL)
Nios II 9.1 Software Build Tools for Eclipse
Nios II 9.1 Uninstall, Repair or Modify

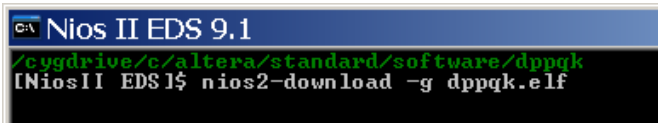
- Run the following command in the first shell to capture all output from on the JTAG UART and redirect to a file. Since this is binary data, the terminal may exit if it receives data representing CTRL-

Copyright © Quantum Leaps, LLC. All Rights Reserved.

D. Luckily there's a flag for that. (If the nios2-terminal command fails, you might have to close eclipse. (It might still be connected to the NEEK via the JTAG.)



- In the other Command Shell download and execute the application:



- Stop nios2-terminal after several seconds.
- Then execute the QSPY host application using the `-f` file option to read the data from the generated file.

```
qspy -f output
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

4 The Vanilla QP Port

The “vanilla” port shows how to use the QP framework on a Nios-II-based system without any underlying multitasking kernel. In the Nios II environment, you never really program to the “bare metal” hardware, because you always use the Altera HAL (Hardware Abstraction Layer) generated for you by the Nios II IDE.

In the “vanilla” version of the QP, the only component requiring platform-specific porting is the QF. The other two components: QEP and QS require merely recompilation and will not be discussed here. Obviously, with the vanilla port you’re not using the QK component. In case of AVR, the “vanilla” QF port is very similar to the generic “vanilla” port described in Chapter 9 of [PSiCC2].

4.1 The qf_port.h Header File

The QF header file for the Nios II, vanilla port, gcc compiler is located in `<qp>\¬ports\¬nios2\¬vanilla\¬gnu\¬qf_port.h`. This header file specifies the interrupt locking/¬unlocking policy (QF critical section) as well as the sizes of various QF objects.

4.1.1 The QF Critical Section

This QF port uses the interrupt locking policy of saving and restoring the interrupt status and described in Chapter 7 of [PSiCC2]. This policy allows for nesting critical sections, where the interrupts status is preserved across the critical section in a temporary stack variable. In other words, upon the exit from a critical section the interrupts are actually unlocked in the `QF_INT_UNLOCK()` macro only if they were unlocked before the matching `QF_INT_LOCK()` macro. Conversely, interrupts will remain locked after the `QF_INT_UNLOCK()` macro if they were locked before the matching `QF_INT_LOCK()` macro.

The critical section in QF is defined as follows:

Listing 2 The QF critical section defined in qf_port.h.

```
                                          /* QF interrupt disable/enable */
(1) #define QF_INT_DISABLE()              ((void)alt_irq_disable_all())
(2) #define QF_INT_ENABLE()               alt_irq_enable_all(0xFFFFFFFFUL)

                                           /* QF critical section entry/exit */
(3) #define QF_CRIT_STAT_TYPE             alt_irq_context
(4) #define QF_CRIT_ENTRY(stat_)          ((stat_) = alt_irq_disable_all())
(5) #define QF_CRIT_EXIT(stat_)           alt_irq_enable_all(stat_)

    #include "sys/alt_irq.h"
```

(1) The unconditional interrupt disable macro resolves to an inline function `alt_irq_disable_all()`, which is defined inline in the header file `sys/alt_irq.h`. The return value from this function is ignored in this macro.

(2) The unconditional interrupt enable macro resolves to an inline function `alt_irq_enable_all(0xFFFFFFFFUL)`, which enables all interrupts and is defined inline in the header file `sys/alt_irq.h`.

(3) The `QF_CRIT_STAT_TYPE` is defined, which means that critical section policy of “saving and restoring the interrupt status” is applied. In this method, the original Nios II Status Register is saved in the variable of the type `alt_irq_context` declared in the file `sys/alt_irq.h`.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

(4) The critical section entry macro resolves to an inline function call that returns the original Nios II Status Register and saves it in the argument `key_`. The function `alt_irq_disable_all()` is defined inline in the header file `sys/alt_irq.h`.

(6) The critical section exit macro resolves to an inline function call that restores the original Nios II Status Register provided in the argument key_. The function `alt_irq_enable_all()` is defined inline in the header file `sys/alt_irq.h`.

NOTE: The name of the function `alt_irq_enable_all()` might be misleading. The function actually only enables these interrupts that correspond to the 1-bits in the argument `key_`.

4.2 ISRs in the Non-preemptive "Vanilla" Configuration

As described in the "Nios II Software Developer's Book" [Altera HAL], the Altera HAL provides the low-level interrupt "wrapper" function `alt_irq_handler()`. This function is called directly from the interrupt vector and calls other registered interrupt handlers for each activated interrupt.

The "vanilla" QP port works just fine with the standard HAL interrupt implementation.

4.2.1 Registering Interrupt Handlers

The interrupt handlers are normal C functions that you register with the HAL function `alt_irq_register()`. You need to register an interrupt handler function for every interrupt that you want to handle.

4.2.2 The Time-Tick Interrupt

In the QP port, you always need to provide a time source that periodically invokes the `QF_tick()` function (`QF::tick()` in C++) to handle the armed Time Events (see Chapter 7 in [PSiCC2]).

The time-tick interrupt is already registered and used by the HAL, so you cannot override it and you don't need to register it. You can, however, define a "hook" to add some more processing to the time-tick interrupt. In this time-tick hook you call the QF-tick processing.

Listing 3 The HAL hooks for the "Vanilla" QP port defined in the file <qp>\ports\nios2\vanilla\gnu\os\alt_hooks.h.

```
#ifndef __ALT_HOOKS_H__
#define __ALT_HOOKS_H__

#define ALT_OS_TIME_TICK()  QF_tick_()
#define ALT_OS_INIT()       ((void)0)
#define ALT_OS_STOP()       ((void)0)

#define ALT_OS_INT_ENTER()  ((void)0)
#define ALT_OS_INT_EXIT()   ((void)0)

void QF_tick_(void);

#endif                                                  // __ALT_HOOKS_H__
```

4.2.3 Optional Nesting of Interrupts

The Altera HAL offers the option of nesting interrupts. You achieve interrupt nesting re-enabling interrupts inside the interrupt handler. The HAL provides a pair of inline functions to be used for this purpose. The

Copyright © Quantum Leaps, LLC. All Rights Reserved.

`QF_tick_()` handler in the C++ port (`<qp>\ports\nios2\vanilla\gnu\src\qf_port.cpp`) provides an example how to use this technique.

Listing 4 Allowing interrupt nesting by using a pair of HA: functions `alt_irq_interruptible()` and `alt_irq_non_interruptible()`.

```
extern "C" void QF_tick_(void) {
    /* enable lower-priority interrupts if the SYS_CLK_TIMER_IRQ is not
    * already the highest-priority IRQ in the system */
#if (SYS_CLK_TIMER_IRQ > 0)
    alt_u32 prio_mask = alt_irq_interruptible(SYS_CLK_TIMER_IRQ);
#endif

    QF::TICK((void *)0);

#if (SYS_CLK_TIMER_IRQ > 0)
    alt_irq_non_interruptible(prio_mask);
#endif
}
```

NOTE: The argument of the function `alt_irq_interruptible()` is the priority level of the currently serviced interrupt. This function re-enables all interrupts above this priority level, while keeping all interrupts with lower or equal priority locked.

NOTE: It is absolutely essential to match every call to the `alt_irq_interruptible()` function with the call to `alt_irq_non_interruptible()`. A failure to call `alt_irq_non_interruptible()` might result in

4.3 Idle Loop Customization in QF_onIdle()

As described in Chapter 7 of [PSiCc2], the “vanilla” port uses the non-preemptive scheduler built into QF. If no events are available, the non-preemptive scheduler invokes the platform-specific callback function `QF_onIdle()`, which you can use to save CPU power, or perform any other “idle” processing (such as Quantum Spy software trace output).

NOTE: The idle callback `QF_onIdle()` must be invoked with interrupts disabled, because the idle condition can be changed by any interrupt that posts events to event queues. `QF_onIdle()` **must** internally enable interrupts, ideally atomically with putting the CPU to the power-saving mode.

The following listing shows the implementation of `QF_onIdle()` from the `bsp.c` file in the DPP example.

```
void QF_onIdle() {                      // NOTE: called with interrupts disabled
    ... // handle the Q_SPY output...
    QF_INT_ENABLE();
}
```

In the QS configuration (`Q_SPY` macro defined) `QF_onIdle()` performs the output of the QS trace data. The upcoming Section 6 discusses the QS output in detail.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

4.4 Assertion Handling Policy in Q_onAssert()

As described in Chapter 6 of [PSiCC2], all QP components use internally assertions to detect errors in the way application is using the QP services. You need to define how the application reacts in case of assertion failure by providing the callback function `Q_onAssert()`. Typically, you would put the system in a fail-safe state and try to reset. It is also a good idea to log some information as to where the assertion failed.

The following code fragment shows the `Q_onAssert()` callback for Nios II. The function simply lights up the LED-7, disables all interrupts and enters an endless loop. This policy is only adequate during the software development, but **must** be changed for production release.

> **NOTE**: This implementation is **not** suitable for the release version of your product. You need to re-implement `Q_onAssert()` in the release version. Typically, you want to put the system in a fail-safe state followed by the reset.

```
void Q_onAssert(char const Q_ROM * const Q_ROM_VAR file, int line) {
    LED_ON(7);
    QF_INT_DISABLE();                          /* disable all interrupts */
    for (;;) {   /* NOTE: replace this endless loop in the production code! */
    }
}
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

5 The QK Port

The QK ports show how to use Quantum Platform on the Nios II processor with the Quantum Kernel (QK), which is a very lightweight, **preemptive, priority-based kernel** designed specifically for QF (see Chapter 10 in [PSiCC2]). You should consider the QK port if your application requires deterministic, real-time performance and also the application can benefit from decoupling higher-priority tasks from lower-priority tasks in the time domain.

One of the biggest advantages of QK is that porting QK to a new microprocessor is very easy. In fact the QK port to Nios II is almost identical to the simplest "vanilla" port described before. The main difference between the two ports, which is visible at the application level, is that you provide the QK-specific interrupt entry and exit code via the HAL hooks `ALT_OS_INT_ENTER()` and `ALT_OS_INT_EXIT()`, respectively. The other slight difference is that you customize the idle loop processing in a different way than in the "vanilla" port. This section focuses only on the differences from the "vanilla" port.

5.1 The qk_port.h Header File

The QK header file for the Nios II, gcc compiler is located in `<qp>\ports\nios2\qk\gnu\qk_port.h`. This file specifies the interrupt enabling/disabling policy (QK critical section). For compatibility with the Altera HAL, the interrupt entry/exit hooks are defined in the file `<qp>\ports\nios2\qk\gnu\os\-alt_hooks.h`.

5.1.1 The QK Critical Section

The QK port uses the same simple critical section described in the "vanilla" port.

5.1.2 The Thread-Local Storage for Newlib

The Altera Nios II development environment uses Newlib as the standard C runtime library. Newlib's facilities are reentrant, but only when properly integrated into a multithreaded environment. Because QK is a **preemptive** kernel, care must be taken to preserve the reentrant character of Newlib.

The Newlib's reentrancy is based on using the "Thead-Local-Storage" (TLS) concept, which is not quite obvious at first glance, so here is a short description how it works (see ESD article "Embedding with GNU: Newlib" [Gatliff 03]). Once you know the details, it will be clear how to make sure you set it up properly in your system.

Newlib declares one `_reent` structure and aims the global `_impure_ptr` pointer at it during initialization, so everything starts out correctly for situations where only one thread of execution is in the library at a time. To facilitate multiple contexts, you must take two additional steps: you must provide one `_reent` structure for **each** execution thread (active object in QF), and you must move `_impure_ptr` between these structures during context switches.

QK supports the TLS concept (see Section 10.4.2 in [PSiCC2]) and provides a context-switch hook `QK_TLS()`, which is invoked every time a different task priority is processed. The following code fragment from `qk_port.h` defines the macro `QK_TLS()` for re-assigning the Newlib's `_impure_ptr` during context switches:

```
                                        /* QK thread-local storage for NewLib */
#define QK_TLS(act_) \
    if ((act_)->thread != (void *)0) { \
        _impure_ptr = (struct _reent *)(act_)->thread; \
    } else ((void)0)
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

While the `QK_TLS()` macro will move the `_impure_ptr`, you are responsible for allocating the `_reent` structure in each active object that actually uses the Newlib facilities. In other words, **you** have the option of not allocating the `_reent` structure and not performing the `_impure_ptr` re-assignment for those active objects that don't use the Newlib.

The DPP sample application for QK provides an example of using the TLS for `Philosopher` active objects, but not for the `Table` active object.

The Philosopher active objects use the TLS as follows (see file `<qp>\examples\nios2\qk\gnu\dpp-qk_cycloneIII_3c25_niosII\philo.c`). First the `struct _reent tls_` is declared as data member of the `Philo` class:

```
typedef struct PhiloTag {
    QActive super;
    struct _reent tls;                   /* thread-local storage for NewLib */
    QTimeEvt timeEvt;                   /* for timing out thining or eating */
} Philo;
```

Next, the pointer to the TLS storage is passed to the framework during the invocation of the `QActivre_start()` function:

```
void Philo_start(uint8_t n,
                 uint8_t p, QEvent const *qSto[], uint32_t qLen)
{
    Philo *me = &l_philo[n];
    Philo_ctor(me);                                        /* instantiate */

    _impure_ptr = &me->tls;   /* initialize thread-local storage for NewLib */
    _REENT_INIT_PTR(_impure_ptr);

    QActive_start((QActive *)me, p, qSto, qLen,
                  &me->tls, (uint8_t)0,
                  (QEvent *)0);
}
```

In contrast, the `Table` active object does not use the TLS, and simply passes the `NULL` pointer to the `QActive_start()` method (see `<qp>\examples\nios2\qk\gnu\dpp-qk_cycloneIII_3c25_niosII\table.c`).

5.2 The alt_hooks.h Header File

The Altera HAL provides a set of "hooks" to extend the basic functionality of the HAL. These hooks are defined as macros in the header file `<qp>\ports\nios2\qk\gnu\os\alt_hooks.h`. Here is this header file for the QK port:

```
#ifndef __ALT_HOOKS_H__
#define __ALT_HOOKS_H__

#define ALT_OS_TIME_TICK()  QF_tick_()
#define ALT_OS_INIT()       ((void)0)
#define ALT_OS_STOP()       ((void)0)

#define ALT_OS_INT_ENTER()  (++QK_intNest_)
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
#define ALT_OS_INT_EXIT()   \
    if ((--QK_intNest_) == (alt_u8)0) { \
        QK_scheduleExt_(0xFFFFFFFFUL); \
    } else ((void)0)

extern alt_u8 QK_intNest_;
void QF_tick_(void);
void QK_scheduleExt_(alt_irq_context intUnlockKey);

#endif                                                  /* __ALT_HOOKS_H__ */
```

The `ALT_OS_TIME_TICK()` hook has been already discussed in the "Vanilla" port. The QK port uses the hook in an identical way.

5.2.1 *The QK-Specific Interrupt Entry and Exit*

The most important difference from the "Vanilla" port is definition of the HAL hooks for QK-specific interrupt entry and exit. The macro `ALT_OS_INT_ENTER()` saves the current QK priority into the automatic variable `pin_` followed by raising the QK priority to the interrupt level `0xFF` (the maximum). The macro `ALT_OS_INT_ENTER()` increments the QK interrupt nesting level, which informs the kernel not to perform synchronous context switches when called inside ISRs. The matching `ALT_OS_INT_EXIT()` restores the original QK priority and invokes the QK scheduler passing to it the interrupt unlock key of `0xFFFFFFFF`, which unlocks all Nios II interrupts.

5.3 The Heap and Environment Mutexes

The Altera HAL provides also ways to protect shared facilities such as the heap and environment variables in the multitasking environment. The heap protection is provided in the file `<qp>\ports\nios2\qk\gnu\src\alt_malloc_lock.c` and the environment protection is provided in the file `<qp>\ports\nios2\qk\gnu\src\alt_env_lock.c`, respectively.

Both files use the same mutual exclusion mechanism, which in case of the preemptive QK is the priority-ceiling mutex (see Section 10.4.1 in [PSiCC2]). Due to the signature to the HAL functions (`__malloc_lock()` and `__env_lock()`, respectively), the QK mutex is not used as defined in the QK interface, but rather a small stack of priorities is maintained to allow nesting of the locks.

The priority ceilings for the malloc lock and env lock are represented as global variables `ALT_malloc_ceiling` and `ALT_env_ceiling`, respectively. Both ceilings are initialized to lock the QK scheduler completely (at the level of `QF_MAX_ACTIVE + 1`), but you can lower the ceilings and let the higher-priority active objects preempt any malloc or env access. Obviously, the higher-priority active objects cannot access malloc or env.

> **NOTE**: It is highly recommended **not** to use the heap or environment variables in high-priority active objects. In this case, you should lower the malloc and env priority ceilings (`ALT_malloc_ceiling` and `ALT_env_ceiling`, respectively) to the priority of the active objects that actually use these facilities. This way you can avoid any negative impact on the timing of the active objects running above the priority ceilings.

5.4 Idle Loop Customization in QK_onIdle()

As described in Chapter 7 of [PSiCC2], the QK idle loop executes only when there are no events to process. The QK allows you to customize the idle loop processing by means of the callback `QK_onIdle()`, which is invoked by every pass through the QK idle loop. You can define the platform-

Copyright © Quantum Leaps, LLC. All Rights Reserved.

specific callback function `QK_onIdle()` to save CPU power, or perform any other "idle" processing (such as Quantum Spy software trace output).

> **NOTE:** The idle callback `QK_onIdle()` is invoked with interrupts unlocked (which is in contrast to `QF_onIdle()` that is invoked with interrupts locked).

In the Nios II port, the callback `QK_onIdle()` can be empty (unlocking the interrupts is not needed since the interrupts are already unlocked). In the DPP example, the `QK_onIdle()` callback is used to output the QS software tracing data, as described in the following section.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

6 QS Software Tracing

Quantum Spy (QS) is a software tracing facility built into all QP components and also available to the Application code. QS allows you to gain unprecedented visibility into your application by selectively logging almost all interesting events occurring within state machines, the framework, the kernel, and your application code. QS software tracing is minimally intrusive, offers precise time-stamping, sophisticated runtime filtering of events, and good data compression (see Chapter 11 in [PSiCC2]).

QS can be configured to send the real-time data using any data link available in the target device. The Nios II design for the NEEK does not have a UART. However, the JTAG UART is available and it is used to output the QS trace data (see also Section 3.7). This provided code contains also the option to use the Nios II UART (which is commented out). The QS platform-dependent implementation is located in the file `bsp.c` and looks as follows:

Listing 5 QSpy implementation to send data out of the JTAG UART or UART.

```
 (1) #ifdef Q_SPY

(2a) //#include "altera_avalon_uart_regs.h"              // uncomment for UART
(2b) #include "altera_avalon_jtag_uart_regs.h"     // uncomment for JTAG UART
 (3) #include "altera_avalon_timer_regs.h"

 (4) #define QS_BUF_SIZE        (1024*4)

 (5) uint8_t QS_onStartup(void const *arg) {
 (6)    static uint8_t qsBuf[QS_BUF_SIZE];               // buffer for Quantum Spy

 (7)    QS_initBuf(qsBuf, sizeof(qsBuf));

        // Use the maximum period for the 32-bit timer
 (8)    IOWR_ALTERA_AVALON_TIMER_PERIODL(HIGH_RES_TIMER_BASE, 0xFFFF);
 (9)    IOWR_ALTERA_AVALON_TIMER_PERIODH(HIGH_RES_TIMER_BASE, 0xFFFF);
        // Start the timer running continuously
(10)    IOWR_ALTERA_AVALON_TIMER_CONTROL(HIGH_RES_TIMER_BASE,
                                         ALTERA_AVALON_TIMER_CONTROL_CONT_MSK
                                         | ALTERA_AVALON_TIMER_CONTROL_START_MSK);

        // Disable receive and transmit interrupts for UART1
        // NOTE: baud rate is set by the hardware configuration
(11)    //IOWR_ALTERA_AVALON_UART_CONTROL(UART1_BASE, 0);    // uncomment for UART


                                                  /* setup the QS filters... */
         QS_FILTER_ON(QS_ALL_RECORDS);

     //    QS_FILTER_OFF(QS_QEP_STATE_EMPTY);
     //    QS_FILTER_OFF(QS_QEP_STATE_ENTRY);
     //    QS_FILTER_OFF(QS_QEP_STATE_EXIT);
     //    QS_FILTER_OFF(QS_QEP_STATE_INIT);
     //    QS_FILTER_OFF(QS_QEP_INIT_TRAN);
     //    QS_FILTER_OFF(QS_QEP_INTERN_TRAN);
     //    QS_FILTER_OFF(QS_QEP_TRAN);
     //    QS_FILTER_OFF(QS_QEP_IGNORED);
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
        QS_FILTER_OFF(QS_QF_ACTIVE_ADD);
        QS_FILTER_OFF(QS_QF_ACTIVE_REMOVE);
        QS_FILTER_OFF(QS_QF_ACTIVE_SUBSCRIBE);
        QS_FILTER_OFF(QS_QF_ACTIVE_UNSUBSCRIBE);
        QS_FILTER_OFF(QS_QF_ACTIVE_POST_FIFO);
        QS_FILTER_OFF(QS_QF_ACTIVE_POST_LIFO);
        QS_FILTER_OFF(QS_QF_ACTIVE_GET);
        QS_FILTER_OFF(QS_QF_ACTIVE_GET_LAST);
        . . .

        return (uint8_t)1;               // indicate successfull QS initialization
    }
    //............................................................................
(12) void QS_onCleanup(void) {
     }
    //............................................................................
(13) void QS_onFlush(void) {
        uint16_t b;
(14)    while ((b = QS::getByte()) != QS_EOD) {   // next QS trace byte available?
(15a)       //while ((IORD_ALTERA_AVALON_UART_STATUS(UART1_BASE)       // for UART
            //       & ALTERA_AVALON_UART_STATUS_TRDY_MSK) == 0)
(15b)       while ((IORD_ALTERA_AVALON_JTAG_UART_CONTROL(JTAG_UART_BASE) // JTAG
                 & ALTERA_AVALON_JTAG_UART_CONTROL_WSPACE_MSK ) == 0 )
            {}
                                        /* write the byte to the TXDATA regiser */
(16a)       //IOWR_ALTERA_AVALON_UART_TXDATA(UART1_BASE, (uint8_t)b);  // for UART
(16b)       IOWR_ALTERA_AVALON_JTAG_UART_DATA( JTAG_UART_BASE, (uint8_t)b); //JTAG
        }
    }
    //............................................................................
    // NOTE: onGetTime is invoked within a critical section (inetrrupts disabled)
(17) QSTimeCtr QSonGetTime(void) {
        // latch the current timer value to the SNAPL and SNAPH registers
(18)    IOWR_ALTERA_AVALON_TIMER_SNAPL(HIGH_RES_TIMER_BASE, 0);
        // read the snapshot from the SNAPL and SNAPH. Invert the timer value
        // to make it a count up timer.
(19)    return ~((IORD_ALTERA_AVALON_TIMER_SNAPH(HIGH_RES_TIMER_BASE) << 16)
                 | IORD_ALTERA_AVALON_TIMER_SNAPL(HIGH_RES_TIMER_BASE));
     }
     #endif                                                          // Q_SPY


(20) void QF_onIdle(alt_irq_context intCtx) {
     #ifdef Q_SPY                        // use the idle cycles for QS transmission
(21)    QS_INT_UNLOCK(intCtx);

(22a)    //while ((IORD_ALTERA_AVALON_UART_STATUS(UART1_BASE) // uncomment for UART
         //       & ALTERA_AVALON_UART_STATUS_TRDY_MSK) != 0)
(22b)    while((IORD_ALTERA_AVALON_JTAG_UART_CONTROL(JTAG_UART_BASE) // JTAG UART
              & ALTERA_AVALON_JTAG_UART_CONTROL_WSPACE_MSK ) != 0 )
         {
(23)        QS_INT_LOCK(intCtx);
(24)        uint16_t b = QS::getByte();
(25)        QS_INT_UNLOCK(intCtx);
(26)        if (b != QS_EOD) {                    // is the QS buffer NOT empty?
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
                                                  // write the byte to the TXDATA regiser
(27a)           //IOWR_ALTERA_AVALON_UART_TXDATA(UART1_BASE, (uint8_t)b);
(27b)           IOWR_ALTERA_AVALON_JTAG_UART_DATA(JTAG_UART_BASE, (uint8_t)b);
            }
            else {
(28)            break;            // the QS buffer is empty, break out of the loop
            }
        }
    #else
(28)    QF_INT_UNLOCK(intCtx);
    #endif                                                              // Q_SPY
    }
```

(1) The QS instrumentation is enabled only when the macro `Q_SPY` is defined
(2-3) The QS instrumentation uses both the Avalon UART and Avalon Timer hardware.
(4) This macro defines the size of the QS buffer size (here 4KB)
(5) The `QS_onStartup()` callback performs the initialization of QS
(6) You must allocate the QS trace buffer of adequate size
(7) You always need to call `QS_initBuf()` from `QS_onStartup()` to initialize the trace buffer
(8-9) The timestamp timer is initialized to count from the maximum
(10) The timestamp timer is initialized to count continuously
(11) UART1 is configured not to generate any interrupts.

NOTE: A fixed baud rate UART is assumed (the standard configuration). See also Altera data sheet "UART Core with Avalon Interface".

(12) The `QS_onCleanup()` callback performs the cleanup of QS. Here nothing needs to be done.
(13-16) The `QS_onFlush()` callback flushes the QS trace buffer to the host. Typically, the function busy-waits for the transfer to complete. It is only used in the initialization phase for sending the QS dictionary records to the host (see Chapter 11 in [PSiCC2])
(17) The `QS_onGetTime()` callback provides the time-stamp to the QS trace records. The QS time-stamping implementation uses the high-resolution hardware Timer configured and started in steps 8-10.
(18) The timer value is atomically latched into the pair of 16-bit `SNAPL` and `SNAPH` registers.
(19) The timestamp value is build from the two 16-bit halves, and additionally the value is inverted to make it an up-counter (the hardware Timer counts down).
(20) The actual QS output is performed in the idle callback (`QF_onIdle()` for the "Vanilla" port, and `QK_onIdle()` for the QK port).
(21) In the `QF_onIdle()` callback the interrupts are first unlocked.
(22) The while loop checks if the UART1 TXDATA register is empty.
(23-25) If so, the next byte is retrieved from the QS buffer in the critical section.
(26-27) Only if the QS buffer is NOT empty, the byte is copied to the TXDATA register.
(28) Otherwise the QS buffer is empty, and we break out of the loop and return from the idle callback.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

7 Related Documents and References

Document	Location
[PSiCC2] "Practical UML Statecharts in C/C++, Second Edition", Miro Samek, Newnes, 2008	Available from most online book retailers, such as amazon.com. See also: http://www.state-machine.com/psicc2.htm
[QP 08] "QP Reference Manual", Quantum Leaps, LLC, 2008	http://www.state-machine.com/doxygen/qpn/
[QL AN-Directory 07] "Application Note: QP Directory Structure", Quantum Leaps, LLC, 2007	http://www.state-machine.com/doc/AN_QP_Directory_Structure.pdf
[QL AN-DPP 08] "Application Note: Dining Philosopher Problem Application", Quantum Leaps, LLC, 2008	http://www.state-machine.com/doc/AN_DPP.pdf
[NiosII SDH] "Nios II Software Developer's Handbook", Altera 2009	Altera document NII5V2-9.1 http://www.altera.com/literature/hb/nios2/-n2sw_nii5v2.pdf
[NEEK 3C25] Standard Hardware Design, NEEK Cyclone III Info, Altera 2009	http://www.altera.com/support/examples/nios2/exm-std_nios2_hardware.html#neek_cycloneIII
[Gatliff 03] "Embedding with GNU: newlib", Bill Gatliff, Embedded Systems Design, October 2003.	http://www.embedded.com/columns/15201376
[Samek 07a] "Using Low-Power Modes in Foreground/Background Systems", Miro Samek, Embedded System Design, October 2007	http://www.embedded.com/design/202103425

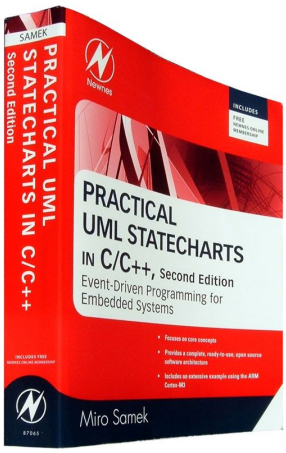
Copyright © Quantum Leaps, LLC. All Rights Reserved.

8 Contact Information

Quantum Leaps, LLC
103 Cobble Ridge Drive
Chapel Hill, NC 27516
USA

+1 866 450 LEAP (toll free, USA only)
+1 919 869-2998 (FAX)

e-mail: info@quantum-leaps.com
WEB : http://www.quantum-leaps.com
http://www.state-machine.com



"*Practical UML Statecharts in C/C++, Second Edition: Event Driven Programming for Embedded Systems*",
by Miro Samek,
Newnes, 2008

Copyright © Quantum Leaps, LLC. All Rights Reserved.